



A behavior-driven approach for specifying and testing user requirements in interactive systems

Thiago Rocha Silva

► To cite this version:

Thiago Rocha Silva. A behavior-driven approach for specifying and testing user requirements in interactive systems. Artificial Intelligence [cs.AI]. Université Paul Sabatier - Toulouse III, 2018. English. NNT : 2018TOU30075 . tel-02129355

HAL Id: tel-02129355

<https://tel.archives-ouvertes.fr/tel-02129355>

Submitted on 14 May 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



THÈSE

En vue de l'obtention du DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par l'Université Toulouse 3 - Paul Sabatier

Présentée et soutenue par
Thiago ROCHA SILVA

Le 17 septembre 2018

A Behavior-Driven Approach for Specifying and Testing User Requirements in Interactive Systems

Ecole doctorale : **EDMITT - Ecole Doctorale Mathématiques, Informatique et
Télécommunications de Toulouse**

Spécialité :

Unité de recherche :

IRIT : Institut de Recherche en Informatique de Toulouse

Thèse dirigée par
Marco Antonio ALBA WINCKLER

Jury

M. Jean VANDERDONCKT, Rapporteur
Mme Káthia MARÇAL DE OLIVEIRA, Rapporteur
Mme Sophie DUPUY-CHESSA, Examinateur
M. Philippe PALANQUE, Examinateur
Mme Célia MARTINIE, Examinateur
M. Marco Antonio ALBA WINCKLER, Directeur de thèse

A Behavior-Driven Approach for Specifying and Testing User Requirements in Interactive Systems

PhD Thesis

Thiago Rocha Silva

Advisor: Prof. Marco Winckler, PhD.

Une approche dirigée par le comportement pour la spécification et le test des exigences utilisateur dans les systèmes interactifs

Thèse de doctorat

Thiago Rocha Silva

Directeur de thèse : Prof. Marco Winckler, PhD.

To my beautiful and lovely wife, Michele.

Acknowledgments

First of all, I would like to thank my advisor, Prof. Marco Winckler, for having accepted to receive me here in Toulouse as his student and having given me all the support, encouragement and incentive to pursue this thesis. Marco, your example inspired me, and your ideas and comments have taken this thesis to another level. I would also like to express my gratitude to you for having introduced us to the French way of life, and for always being our guarantor in everything we needed. Um mais que sincero, muito obrigado!

I would also like to thank my rapporteurs, Prof. Jean Vanderdonckt and Káthia Marçal, for accepting the invitation to review this work and for the precious and precise comments and suggestions. They helped me to see many important points to improve and opened my mind to several other possibilities to continue this research. Jean, I really admire your work and the brilliant mind you have, I feel so honored you took the time to revise my work and having been here in Toulouse to take part in my jury. Káthia, eu não poderia deixar de registrar um agradecimento especial a você (e em português que você tanto sente falta ☺), por tão detalhada revisão e comentários cruciais especialmente sobre a ontologia e sobre os estudos de caso. Adoramos te conhecer! I also thank all the jury, Prof. Philippe Palanque, Prof. Sophie Dupuy-Chessa and Célia Martinie, for all your questions, suggestions and for accepting to take part in this jury even with a tight schedule.

A special thanks to the whole ICS team, past and current members, for having welcomed me and Michele so well and included us in all your events during these four years. I'm very proud of being part of such a high-level and internationally recognized team. Thanks Phil, for being such a funny, friendly, and brilliant boss; Célia for being always so friendly and available; Didier for being an example of kindness, you helped me a lot with technical and practical stuff, making the daily tasks much easier; David, Regina, Arnaud, Camille, Martin, Elodie, Alexandre(s), Racim, Dimitri, François, Guillaume, a big thanks for all these years together. A special thanks to Jean-Luc and Pedro Valente, who collaborated with me in some topics of this thesis. I also thank the IRIT and EDEMITT administrative staff, especially Arnaude and the financial department team (Lorène, Véronique, Léonor and Matthew) for participating in the case study, Chantal Morand for having helped us so much in our arrival, Agnès and Martine for always being so kind and helpful. Un grand merci à vous tous! Agradeço ao Prof. Marcelo Pimenta (UFRGS) pela disponibilidade em contribuir com o trabalho, foi uma pena não termos conseguido te trazer para o júri. Aos professores da UFMG Clarindo (meu orientador de mestrado), Rodolfo, Raquel e Marco Túlio por me receberem tão bem e contribuírem com valiosos comentários sobre o tema da tese durante a minha última passagem pelo DCC. Aos professores e amigos da Unimontes pelos anos de aprendizado e companheirismo, em especial Guilherme e Chris.

Je remercie à Serge pour les tout premiers cours de Français à Toulouse et à tout(e)s mes ami(e)s du groupe Toulouse-B2: Arash, Javier, Josipa, Marie Eline, Jimena, Ponleu, Flávio, Lívia, Yuri e Nina, merci à vous tous pour les moments amusants et agréables que nous avons passés ensemble. Aos amigos brasileiros em Toulouse: Paulo, Marina, Fábio, Helena, Thaíse, Achilles, Filipe, Lilian (et Loïc, pas brésilien mais presque ☺), muitíssimo obrigado pela companhia e pelos ótimos momentos que partilhamos por aqui. Certamente, a jornada foi muito mais fácil com vocês!

Agradeço muito especialmente a toda minha família e amigos queridos que tanto nos apoiaram nessa jornada e entenderam a nossa ausência durante todo esse período. Muito obrigado, mãe

(Fátima) e pai (João), por sempre me motivarem e insistirem que a educação era o caminho. Eu não teria chegado até aqui sem vocês! Meus queridos irmãos, Lucas e Pedro (também meu afilhado), pelo incentivo e pelo companheirismo. Sei o quanto todos vocês estão orgulhosos dessa conquista. Minha grande amiga-irmã, Carol, assim como o Emerson, o meu muito obrigado pela amizade sincera e pelo amor e carinho de sempre. Obrigado por fazerem de tudo e nos deixarem sempre participar da vida e do crescimento das meninas, mesmo estando de longe. Li, minha prima-amiga-irmã, obrigado por ter compartilhado comigo esse coração do tamanho do mundo e ter sempre cuidado tão bem de mim. Gil, por toda a camaradagem de sempre. Aos meus demais afilhados Lê, Bia e Ju, que tanto alegram os nossos dias e não nos deixam nunca esquecer o quanto vocês são doces e amáveis! Amo muito vocês! Aos meus tios e tias, primos e primas, em especial Mazza, Zilmio, Zelândia, Kênia, Zildete e Alexandre, o meu mais sincero agradecimento pelo amor e carinho e por sempre estarem por perto. Aos meus padrinhos Reis e Lete, obrigado por terem sempre me dado todo o suporte e apoio necessários. À toda família Silva pelo acolhimento, à Ni em especial, que tanto participou da minha criação.

O meu muito obrigado aos meus demais amigos pela torcida e apoio, em especial Aline (de quem acabamos perdendo o casamento por conta do doutorado, aqui vai mais um pedido de desculpas), Marius, Adélia (e Gabriel). Aos grandes amigos da Fabrai/Anhanguera/Una: Jéferson e Marcela (de quem, como padrinhos, também acabamos perdendo o casamento por conta do doutorado, aqui vai mais um pedido de desculpas), Hélio e Jordana, Lindenbergs e Tânias, Dani, Rodrigo, Sandro, Ernani e Helê. À toda a família Mendonça, em especial Káthia, Mendonça, Hugo e Duda que sempre me acolheram com tanto carinho, sobretudo nos momentos em que eu mais precisei.

À toda família Rodrigues que me recebeu de braços tão abertos e sempre foram um grande exemplo de união e harmonia. Edvaldo e Albanita, obrigado por todo o carinho e por me confiarem a Michele ☺. Dani, você foi o nosso alicerce em muitos momentos ao longo desses anos fora, e o meu agradecimento a você vai muito além de um simples obrigado, eu tive a sorte de conhecer a pessoa maravilhosa que você é. Nos desculpe por não estarmos tão presentes como gostaríamos. Erik, obrigado por sempre nos receber tão bem. Dieguinho, você ajudou a deixar os dias do tio Thiago bem mais alegres durante esses anos.

À minha linda e querida esposa Michele, a quem dedico essa tese, por todo amor, suporte, carinho, companheirismo, amizade, doação e coragem. Sem você, meu amor, eu não teria chegado até aqui. O teu apoio incondicional, a tua ajuda com os resultados da tese, tudo isso foi indispensável nessa caminhada. Tenho muita sorte de poder acordar com os teus lindos olhos ao meu lado todas as manhãs. Sei o quanto essa mudança foi difícil para você, sei o quanto você deixou projetos para trás para entrar de cabeça e me acompanhar nesse sonho. Serei eternamente grato a você por isso. Espero que com a realização desse objetivo venham novos desafios; conto com você ao meu lado para cada um deles. Te amo infinitamente!

I thank CAPES and the Brazilian government for believing in this project and fully funding it. Agradeço também ao Serpro pela minha liberação (em especial aos grandes incentivadores Bráulio e Alexandre Barros) e aos colegas da empresa pelos anos de aprendizado até aqui, em especial Lara e Maurílio. À Marta, por todo o suporte administrativo.

Por fim, mas definitivamente não menos importante, agradeço a Deus, à vó Benta e aos meus guias espirituais por estarem sempre comigo, me proporcionarem essa e tantas outras oportunidades e vitórias, nunca me deixarem desistir e me acompanharem mesmo nas mais difíceis caminhadas.

Summary

Part I – Introduction

Chapter 1: Introduction	25
1.1. Context	25
1.2. Challenges	27
1.3. Objectives	28
1.4. Methodological Approach	30
1.5. Thesis' Outline	31
Chapter 2: Background	35
2.1. Methods for Modeling User Requirements for Interactive Systems	35
2.1.1. User Stories and Scenario-Based Design	35
2.1.2. Task Analysis and Modeling	40
2.1.3. User Interface Prototyping	44
2.1.4. User Interfaces and Task-Based Development	46
2.2. Methods for Evaluating User Requirements	47
2.2.1. Functional Testing	48
2.2.2. GUI Testing	50
2.2.3. Artifacts Inspection and Requirements Traceability	50
2.3. Software Development Processes	51
2.3.1. Agile Methods	52
2.3.2. Behavior-Driven Development	55
2.4. Conclusion	56
2.5. Resultant Publications	56

Part II – Contribution

Chapter 3: A Scenario-Based Approach for Multi-Artifact Testing	61
3.1. Rationale for a Scenario-Based Approach	61
3.1.1. Target Stakeholders	65
3.2. Multiple Views of the Approach	65
3.2.1. Architectural View	66
3.2.2. Workflow View	67
3.2.3. Alternatives for Performing the Approach	69
3.3. A Case Study in a Nutshell	71

3.3.1.	Writing Testable User Stories	72
3.3.2.	Adding Testing Scenarios	73
3.4.	Strategy for Testing	74
3.5.	Conclusion	75
3.6.	Resultant Publications	77
Chapter 4: Towards an Ontology for Supporting GUI Automated Testing		79
4.1.	Related Approaches	80
4.1.1	Compared Overview	80
4.2.	A Behavior-Based Ontology for Interactive Systems	81
4.2.1	Object Properties	83
4.2.2	Relations	83
4.2.3	Data Properties	84
4.2.4	Platform Concepts	85
4.2.5	UI Elements Concepts	86
4.2.6	State Machine Concepts	91
4.2.7	Scenario-Based Concepts	91
4.2.8	Consistency Checking	95
4.3.	Contributions, Limitations and Perspectives	95
4.4.	Resultant Publications	97
Chapter 5: Modeling and Assessing Task Models		99
5.1.	An Overview of HAMSTERS	100
5.1.1.	Task Types	100
5.1.2.	Operators	102
5.1.3.	Extracting Scenarios	103
5.1.4.	Handling Data	103
5.2.	Modeling User's Tasks	104
5.3.	Assessing User's Tasks	106
5.3.1.	Extracting Scenarios and Formatting User Stories	106
5.3.2.	Elements Mapped for Testing	109
5.3.3.	Implementation	111
5.3.4.	Towards an Alternative to the Extraction of Scenarios	118
5.4.	Conclusion	120
5.5.	Resultant Publications	121
Chapter 6: Modeling and Assessing User Interfaces: From Prototypes to Final UIs		123
6.1.	Starting with Balsamiq Wireframes	125
6.1.1.	Test Implementation	127

6.2.	Using the Ontology to Support the Development of Consistent Prototypes	130
6.3.	Evolving UI Prototypes	133
6.3.1.	Elements Mapped for Testing	135
6.4.	Testing Final User Interfaces	136
6.4.1.	Integrated Tools Architecture	136
6.4.2.	Implementation	137
6.4.3.	Handling Test Data	140
6.5.	Conclusion	142
6.6.	Resultant Publications	143

Part III – Evaluation

Chapter 7: Case Study 1 - Understandability of User Stories	147	
7.1.	Experimental Design	147
7.2.	Methodology	148
7.3.	The Business Narrative	149
7.4.	Participant's Profile	151
7.5.	The Proposed Exercise	151
7.6.	Results	152
7.6.1.	User Stories Writing	153
7.6.2.	Adherence Analyses	156
7.6.3.	Discussion	164
7.7.	Findings and Implications	168
7.7.1.	Threats to Validity	170
7.8.	Conclusion	171
Chapter 8: Case Study II - Assessing User Interface Design Artifacts	173	
8.1.	Case Study Design	173
8.2.	Formatting and Adding New User Stories	175
8.3.	Adding Testing Scenarios	178
8.4.	Modeling and Assessing Task Models	178
8.4.1	Extracting Scenarios from the Task Models	180
8.4.2	Results	182
8.4.3	Types of Inconsistencies Identified	189
8.5.	Modeling and Assessing UI Prototypes	190
8.5.1.	Types of Inconsistencies Identified	202
8.6.	Assessing Final UIs	203
8.6.1.	Types of Inconsistencies Identified	205

8.7.	Results Mapping	214
8.8.	Summary of Main Findings in the Case Study	218
8.9.	Threats to Validity	221
8.10.	Conclusion and Lessons Learned	221

Part IV - Conclusion

Chapter 9: Conclusion	225	
9.1.	Tackled Challenges	226
9.2.	Summary of Contributions	228
9.3.	Summary of Limitations	228
9.4.	Future Research Perspectives	229
9.4.1.	Short Term Perspective	229
9.4.2.	Long Term Perspective	229
9.5.	Full List of Resultant Publications	230
References	233	
Appendix A: Concept Mapping Table	243	
Appendix B: Log of Results - Assessing Task Models	251	
Annex A: Case Study Interview Protocol	267	
Annex B: User Stories Written by the Case Study Participants	275	
Annex C: Transcription of the Interviews	279	

List of Figures

Figure 1. Requirements and artifacts being “photographed” in different phases of the project.	27
Figure 2. An overview of the scenario-based design (SBD) framework (Rosson and Carroll, 2002).	36
Figure 3. The logic model of a user interface (Green, 1985).	45
Figure 20. The Cameleon Reference Framework.	46
Figure 4. The V-model for testing.	49
Figure 5. Simplified versions of waterfall and iterative models.	52
Figure 6. Agile Model Driven Development (AMDD) (Ambler, 2002).	54
Figure 7. “Subway Map” to agile practices (<i>Agile Alliance</i> , 2018).	54
Figure 8. The cycle of permanent evolution of artifacts in iterative processes	62
Figure 9. Modeling business and functional requirements in a scenario-based approach.	63
Figure 10. Conceptual Model for testable requirements.	64
Figure 11. Overall view of the approach.	65
Figure 12. Architectural view of the approach.	66
Figure 13. Workflow view of the approach.	68
Figure 14. Alternatives for performing the approach.	69
Figure 15. The graph of options for performing our approach.	70
Figure 16. Business Process Model for the flight ticket e-ticket domain.	71
Figure 17. Activity of telling User Stories	72
Figure 18. Activity of creating testing scenarios	73
Figure 19. Our strategy for testing.	75
Figure 21. Main classes and their properties in the ontology.	82
Figure 22. Object Properties <i>isComposedBy</i> (left) and <i>isTriggeredBy</i> (right).	83
Figure 23. Left: Data Properties. Right: Data Property “message”.	85
Figure 24. Example of Web and Mobile implementations of a Calendar.	86
Figure 25. Cloud of User Interface (UI) Elements.	87
Figure 26. State Machine Elements and their Individuals.	91
Figure 27. A Transition being represented in the State Machine.	91
Figure 28. Components on the ontology used to specify a behavior.	92
Figure 29. Behavior “chooseRefferingTo”.	92
Figure 30. Results of ontology processing: HermiT (top) and Pellet (bottom).	95
Figure 31. One of the alternatives to perform our approach.	99
Figure 32. Example of Task Properties.	101
Figure 33. Representation of executable and executed tasks during simulation.	103
Figure 34. Example of “Information” and “Data” handling.	103
Figure 35. Activity of creating task models.	104
Figure 36. Mapping BPMN business activities to HAMSTERS user tasks.	105
Figure 37. Activity of formatting User Stories.	106
Figure 38. Scenarios being extracted from task models and then being formatted by the ontology as User Stories.	107
Figure 39. Formatting rule for assessing steps and tasks.	109
Figure 40. Extract of an original (left side) and a resultant (right side) scenario XML files after the process of preformatting.	111
Figure 41. Example of scenario extracted from a task model and its XML source file.	112
Figure 42. Activity of evaluating task models.	112
Figure 43. Checking consistency of tasks between US scenario and scenarios extracted from task models.	112

Figure 44. Testing algorithm for assessing scenarios extracted from task models.....	113
Figure 45. File tree for the implementation of task model assessment.....	116
Figure 46. Flow of calls for running tests on task model scenarios.....	116
Figure 47. “MyTest” class indicating the file “search.story” for running	117
Figure 48. Console after running the User Story “Flight Tickets Search”.....	118
Figure 49. Task model (1), extracted scenario (3), and their respective source files (2 and 4).119	
Figure 50. Flow of activities to get scenarios for testing.....	120
Figure 51. Another alternative for performing our approach.	123
Figure 52. Balsamiq handmade-style UI elements.....	125
Figure 53. Activity of prototyping UIs.	125
Figure 54. Sketch for the User Story “Flight Tickets Search” built from the scenario “One-Way Tickets Search”.....	126
Figure 55. Activity of evaluating UI prototypes.	127
Figure 56. Button “Search” and its XLM source file.	127
Figure 57. Grouped field “Departure Date” and its XLM source file.....	128
Figure 58. Testing algorithm for assessing UI prototypes.	128
Figure 59. “MyTest.java”: class for running tests on Balsamiq prototypes.	129
Figure 60. Flow of calls for running tests on Balsamiq prototypes.	130
Figure 61. PANDA screenshot.....	131
Figure 62. Example of a step split during its parsing.....	132
Figure 63. Properties of a button in the tool PANDA with properties defined by the ontology.	132
Figure 64. Example of results given during a Scenario testing.....	132
Figure 65. Activity of evolving UI prototypes.....	133
Figure 66. The less refined prototype for “Flight Tickets Search” evolving to a more refined one, and then to a final UI.....	134
Figure 67. The “Choose Flights” UI prototype in PANDA.	135
Figure 68. The “Choose Flights” final UI.	135
Figure 69. Activity of evaluating Final UIs.....	136
Figure 70. A 3-module integrated tools architecture.....	136
Figure 71. Flow of components in the proposed architecture.	137
Figure 72. Packages and classes being structured to implement our testing approach.....	137
Figure 73. Parsing a step from a TXT file to a Java method.	138
Figure 74. MyPage Java class.	138
Figure 75. Automated execution of the scenario “Return Tickets Search”.	139
Figure 76. An attempt to select a return date before the departure date.	139
Figure 77. Package tree (on the left) and MyTest class (on the right).	140
Figure 78. Writing a User Story and getting instant feedback of unknown steps.	140
Figure 79. JUnit green/red bar at the left, and JBehave detailed report at the right.	140
Figure 80. Data in Data Provider: (a) data being associated to a variable to be used in the step.	141
Figure 81. Data stored in an XML file: (a) data associated to XML file, (b) reference to dataset.	142
Figure 82. BPMN model for the case study.....	149
Figure 83. Travel Planet system for booking business trips.	150
Figure 84. Structure of a User Story presented to the participants translated to English.	152
Figure 85. Example of a User Story presented to the participants translated to English.....	152
Figure 86. User Story written by P1.....	154
Figure 87. User Story written by P2.....	154
Figure 88. User Story written by P3.....	154

Figure 89. User Story written by P4.....	155
Figure 90. Understandability of Each Statement in the User Story Specification.	164
Figure 92. General Understandability of User Stories (Number of occurrences in each stratum).	165
Figure 91. Understandability in User Story Specification - Narrative (Number of occurrences in each stratum).....	165
Figure 93. Adherence to the Ontology in User Story Specification - Scenario (Number of occurrences in each stratum)......	165
Figure 94. Number of occurrences in each category of adherence problems.	167
Figure 95. Boxplot of each type of adherence problems identified in participants' User Stories.	168
Figure 96. User Story "Flight Tickets Search".	176
Figure 97. User Story "Select a suitable flight".....	177
Figure 98. User Story "Confirm Flight Selection".....	178
Figure 99. Test scenarios for the User Stories.	178
Figure 100. Task Model for Searching Flights using Travel Planet.	179
Figure 101. Task Model for Informing a Flight Leg in Travel Planet.	179
Figure 102. Task Model for Choosing a Flight in Travel Planet.	180
Figure 103. Scenarios extracted to be tested.	182
Figure 104. Results of matching: scenario "Confirm a Flight Selection".	183
Figure 105. Results of matching: scenario "Confirm a Flight Selection (Full Version)".....	187
Figure 106. Results of matching: scenario "Confirm a Flight Selection for a One-Way Trip".	188
Figure 107. Results of matching: scenario "Confirm a Flight Selection for a Multidestination Trip".	189
Figure 108. Results of matching: scenario "Decline a Flight Selection"	189
Figure 109. UI prototype for searching flights (first version).	192
Figure 110. UI prototype for searching flights (revised version after testing).	192
Figure 111. UI prototype for choosing flights (first version).	197
Figure 112. UI prototype for choosing flights (revised version after testing).	197
Figure 113. UI prototype for confirming a booking (first version).	199
Figure 114. UI prototype for confirming a booking (revised version after testing).....	199
Figure 115. UI prototype: Trip Confirmed.....	200
Figure 116. UI prototype: Withdrawing confirmation.....	200
Figure 117. UI prototype: Trip Canceled.	200
Figure 118. UI prototype: Multidestination search.	201
Figure 119. UI prototype: Flight selected.....	201
Figure 120. Final UI for searching flights.	206
Figure 121. Final UI for searching multidestination flights.....	206
Figure 122. Final UI for choosing flights.....	206
Figure 123. Final UI with the selected flights.	210
Figure 124. Final UI for confirming the selected flights.....	210
Figure 125. Final UI: dialog box before canceling.	212
Figure 126. Final UI: trip canceled.....	212

List of Tables

Table 1. Approaches for describing User Stories and Scenarios.....	39
Table 2. Correlation between scenarios in UCD and SE approaches (adapted from (Santoro, 2005)).....	40
Table 3. Target stakeholders of the approach.	65
Table 4. A compared overview between the ontology and other methods and languages.....	81
Table 5. “Relations” as Object Properties in the ontology.....	84
Table 6. Data Properties in the ontology.	85
Table 7. UI Elements in the ontology.....	91
Table 8. Predefined Behaviors described in the ontology.....	95
Table 9. Task types in HAMSTERS.	101
Table 10. Illustration of the operator types within HAMSTERS.....	102
Table 11. The correlation between requirements, tasks and scenarios in UCD and SE approaches for the User Story “Flight Tickets Search”.....	108
Table 12. The correlation between requirements, tasks and scenarios in UCD and SE approaches for the User Story “Select the desired flight”.....	108
Table 13. Task name components construction.....	110
Table 14. Concept mapping for the scenario “One-Way Tickets Search”.	110
Table 15. Checking consistency of tasks between US scenario and scenarios extracted from task models.....	115
Table 16. Example of concept mapping for testing.	135
Table 17. Participant’s Profile.	151
Table 18. User Story Specification – Participant P1.....	159
Table 19. User Story Specification – Participant P2.....	160
Table 20. User Story Specification – Participant P3.....	162
Table 21. User Story Specification – Participant P4.....	163
Table 22. Understandability of Each Statement in the User Story Specification.	164
Table 23. Understandability in User Story Specification - Narrative (Number of occurrences in each stratum).....	165
Table 24. Adherence to the Ontology in User Story Specification - Scenario (Number of occurrences in each stratum).	165
Table 25. Scenario “Confirm a Flight Selection”.....	183
Table 26. Type of inconsistencies identified in scenarios extracted from task models.....	186
Table 27. Scenario “Confirm a Flight Selection (Full Version)”.	187
Table 28. Scenario “Confirm a Flight Selection for a One-Way Trip”.....	188
Table 29. Scenario “Confirm a Flight Selection for a Multidestination Trip”.	188
Table 30. Scenario “Decline a Flight Selection”.....	189
Table 31. Test results in Balsamiq prototypes.	200
Table 32. Test results on the final UI.	213
Table 33. Mapping of the results after testing.	218
Table 34. Main kinds of problems identified in each artifact after testing.....	218

Abstract

In a user-centered design process, artifacts evolve in iterative cycles until they meet user requirements and then become the final product. Every cycle gives the opportunity to revise the design and to introduce new requirements which might affect the artifacts that have been set in former development phases. Keeping the consistency of requirements in such artifacts along the development process is a cumbersome and time-consuming activity, especially if it is done manually. Nowadays, some software development frameworks implement Behavior-Driven Development (BDD) and User Stories as a means of automating the test of interactive systems under construction. Automated testing helps to simulate user's actions on the user interface and therefore check if the system behaves properly and in accordance with the user requirements. However, current tools supporting BDD requires that tests should be written using low-level events and components that only exist when the system is already implemented. As a consequence of such low-level of abstraction, BDD tests can hardly be reused with more abstract artifacts. In order to prevent that tests should be written to every type of artifact, we have investigated the use of ontologies for specifying both requirements and tests once, and then run tests on all artifacts sharing the ontological concepts. The resultant behavior-based ontology we propose herein is therefore aimed at raising the abstraction level while supporting test automation on multiple artifacts. This thesis presents this ontology and an approach based on BDD and User Stories to support the specification and the automated assessment of user requirements on software artifacts along the development process of interactive systems. Two case studies are also presented to validate our approach. The first case study evaluates the understandability of User Stories specifications by a team of Product Owners (POs) from the department in charge of business trips in our institute. With the help of this first case study, we designed a second one to demonstrate how User Stories written using our ontology can be used to assess functional requirements expressed in different artifacts, such as task models, user interface (UI) prototypes, and full-fledged UIs. The results have shown that our approach is able to identify even fine-grained inconsistencies in the mentioned artifacts, allowing establishing a reliable compatibility among different user interface design artifacts.

Keywords: Behavior-Driven Development (BDD), User Stories, Automated Requirements Assessment, Ontological Modeling, Scenario-Based Design, User Interface Design Artifacts.

Résumé

Dans un processus de conception centré sur l'utilisateur, les artefacts évoluent par cycles itératifs jusqu'à ce qu'ils répondent aux exigences des utilisateurs et deviennent ensuite le produit final. Chaque cycle donne l'occasion de réviser la conception et d'introduire de nouvelles exigences qui pourraient affecter les artefacts qui ont été définis dans les phases de développement précédentes. Garder la cohérence des exigences dans tels artefacts tout au long du processus de développement est une activité lourde et longue, surtout si elle est faite manuellement. Actuellement, certains cadres d'applications implémentent le BDD (Développement dirigé par le comportement) et les récits utilisateur comme un moyen d'automatiser le test des systèmes interactifs en construction. Les tests automatisés permettent de simuler les actions de l'utilisateur sur l'interface et, par conséquent, de vérifier si le système se comporte correctement et conformément aux exigences de l'utilisateur. Cependant, les outils actuels supportant BDD requièrent que les tests soient écrits en utilisant des événements de bas niveau et des composants qui n'existent que lorsque le système est déjà implémenté. En conséquence d'un tel bas niveau d'abstraction, les tests BDD peuvent difficilement être réutilisés avec des artefacts plus abstraits. Afin d'éviter que les tests doivent être écrits sur chaque type d'artefact, nous avons étudié l'utilisation des ontologies pour spécifier à la fois les exigences et les tests, puis exécuter des tests dans tous les artefacts partageant les concepts ontologiques. L'ontologie fondée sur le comportement que nous proposons ici vise alors à éléver le niveau d'abstraction tout en supportant l'automatisation de tests dans des multiples artefacts. Cette thèse présente tel ontologie et une approche fondée sur BDD et les récits utilisateur pour soutenir la spécification et l'évaluation automatisée des exigences des utilisateurs dans les artefacts logiciels tout au long du processus de développement des systèmes interactifs. Deux études de cas sont également présentées pour valider notre approche. La première étude de cas évalue la compréhensibilité des spécifications des récits utilisateur par une équipe de propriétaires de produit (POs) du département en charge des voyages d'affaires dans notre institut. À l'aide de cette première étude de cas, nous avons conçu une deuxième étude pour démontrer comment les récits utilisateur rédigés à l'aide de notre ontologie peuvent être utilisés pour évaluer les exigences fonctionnelles exprimées dans des différents artefacts, tels que les modèles de tâche, les prototypes d'interface utilisateur et les interfaces utilisateur à part entière. Les résultats ont montré que notre approche est capable d'identifier même des incohérences à grain fin dans les artefacts mentionnés, permettant d'établir une compatibilité fiable entre les différents artefacts de conception de l'interface utilisateur.

Mots Clés : Développement dirigé par le comportement (BDD), Récits utilisateur, Evaluation automatisée des exigences, Modélisation ontologique, Conception par scénarios, Artefacts de conception d'interface utilisateur.

Part I - Introduction

Chapter 1

Introduction

Summary

This chapter introduces and motivates this thesis, providing along five sections its overall context and problem statement, followed by the set of research challenges we have identified. We also state our aims and objectives towards the solution, as well as the research scope and the methodological approach we have followed. This chapter ends with the thesis' outline.

1.1. Context

Understanding user requirements is critical to the success of interactive systems (Maguire and Bevan, 2002). As a statement of users' expectations and needs about the system, user requirements play a central role in a user-centred design (ISO, 1999). User requirements specifications must express the needs of different stakeholders with different points of view about the system. Being stakeholders anyone who is materially impacted by the outcome of the software solution (Ambler, 2002), such needs may address functional aspects of the system (such as features that the system must provide) or non-functional aspects (such as issues related to performance, usability, scalability, and so on). A stakeholder could be a direct (or end) user, indirect user, manager of users, senior manager, operations staff member, the person who funds the project, support staff member, auditors, the program/portfolio manager, developers working on other systems that integrate or interact with the one under development, or maintenance professionals potentially affected by the development and/or deployment of a software project. To succeed, a software project needs therefore to understand and synthesize their requirements into a cohesive vision (Ambler, 2002).

Each stakeholder has their own requirements, their own vision, and their own priorities. While business people and managers, for example, could be more interested in describing a requirement in a rigid workflow perspective and demand features complying with that, end-users could be more interested in detailing such a requirement in features prioritizing shortcuts or alternative flows in the work process. Such different perspectives about the system behavior can lead to several misunderstandings or conflicting specifications.

The degree of formality in user requirements vary considerably though. Such requirements can be expressed from informal natural language statements until formal object-oriented specifications. The level of formality has a strong impact on the way users and developers can communicate about the requirements. On one hand, informal statements in natural language tend to be easier for users to express their needs, and consequently to understand what is being specified and documented. However, the lack of formalization might allow verbose, incomplete, and ambiguous description of requirements that are difficult to understand and assess. On the other hand, despite largely supporting automation and the system design, formal specifications are difficult for users and non-technical people to understand, which harm their ability to communicate with developers effectively.

In an attempt to formalize the user requirements and how they should be addressed during the implementation, software designers make use of models. Modeling is recognized as a crucial activity to manage the abstraction and the inherent complexity of developing software systems. Several aspects of information, from the macro business goals until the most detailed information about user tasks, are taken into account while modeling. As a consequence, software systems are designed based on several requirements artifacts which model different aspects and different points of view about the system (e.g. business models, use cases, task models, etc.). Artifacts encode a particular interpretation of a problem situation and a particular set of solutions for the perceived problem (De Souza, 2005). They are the means by which the outcomes of modeling activities are registered. Considering that different phases of development require distinct information, artifacts used for modeling tend to be very diverse throughout the development, and ensuring their consistency is quite challenging (Winckler and Palanque, 2012).

Requirements and artifacts are also expected to evolve along the project according to the users' changing perception about their own needs. In iterative processes, the cycle of producing and evolving artifacts permeates all the phases of system development, from requirements and business analysis until software testing. Artifacts are supposed to be kept and maintained even after the software development has finished, once they encompass models describing the record of implemented requirements, which strategies were used to implement the features, how the system architecture has been structured, etc. They are also useful to software maintenance and evolution purposes. Therefore, requirements should be described in a consistent way across the multiple artifacts. Requirements specifications should not, for example, describe a given requirement in a use case that is conflicting with its representation in an activity diagram.

Most of the research intended to ensure some level of consistency between requirements and artifacts is centered on tracing requirements throughout the development process. Requirements traceability is defined as "*the ability to follow the life of a requirement, in both forwards and backwards direction (i.e. from its origins, through its development and specification, to its subsequent deployment and use, and through periods of on-going refinement and iteration in any of these phases)*" (Ramesh *et al.*, 1995). The traceability of requirements and artifacts is usually classified as vertical and horizontal (Ebert, 2011). Vertical traceability describes the relationship between artifacts that can be derived from each other and are in different levels of abstraction, for example, from customer requirements to acceptance test cases. Horizontal traceability, on the other hand, refers to trace the evolution of the same artifact along its lifecycle. The problem of tracing requirements and artifacts has been studied by several authors for decades, and a wide set of commercial tools have been developed to address this problem in various approaches (Nair, De La Vara and Sen, 2013). Nonetheless, proposed solutions to promote vertical traceability between requirements and artifacts can simply identify whether a requirement is present or not in a given artifact, not allowing to effectively test it by checking the consistency and correct representation of such a requirement in a given set of artifacts.

Since long time ago, it is a peaceful argument that providing early assessment is very helpful for detecting errors before making strong commitments with the software implementation (van Megen and Meyerhoff, 1995). Lindstrom (Lindstrom, 1993) declared that failure to trace tests to requirements, for example, is one of the five most effective ways to destroy a project. However, according to Uusitalo *et al.* (Uusitalo *et al.*, 2008), traceability between requirements and tests is rarely maintained in practice. This is caused primarily by failure to update traces when requirements change, due to stringent enforcement of schedules and budgets, as well as difficulties to conduct testing processes through a manual approach. The authors pointed out that in most cases, interviewees in industry longed for better tool support for traceability. Some also

noted that poor quality of requirements was a hindrance to maintaining the traces, since there is no guarantee how well the requirements covered the actual functionality of the product.

1.2. Challenges

Assessing interactive systems is an activity that requires a considerable amount of efforts from development teams. A first challenge for testing is that requirements are not stable along the software development process. Stakeholders introduce new demands or modify the existing ones all along the iterations. This fact makes imperative to retest all the software outcomes in order to ensure that the system remains behaving properly, and the different artifacts remains in accordance with the new requirements introduced and/or modified. Manual tests and software inspections are usually the first approaches to assess these outcomes. However, manually ensuring the consistency of system and artifacts every time a requirement is introduced and/or modified is a discouraging activity for software development teams. Manual tests are extremely time-consuming and highly error-prone. Therefore, promoting automated tests is a key factor to support testing in an ever-changing environment. They allow a reliable and fast assessment of requirements and promote a high availability of tests.

A second challenge for testing requirements is that multiple artifacts with different levels of abstraction might be concerned by the same requirement. Tests therefore should run not only on the final product, but also in the whole set of modeling artifacts to ensure that they represent the same information in a non-ambiguous way, and in accordance with the whole requirements chain. It is indeed a challenge verifying and checking such artifacts while ensuring their consistency with the other components of a requirements specification.

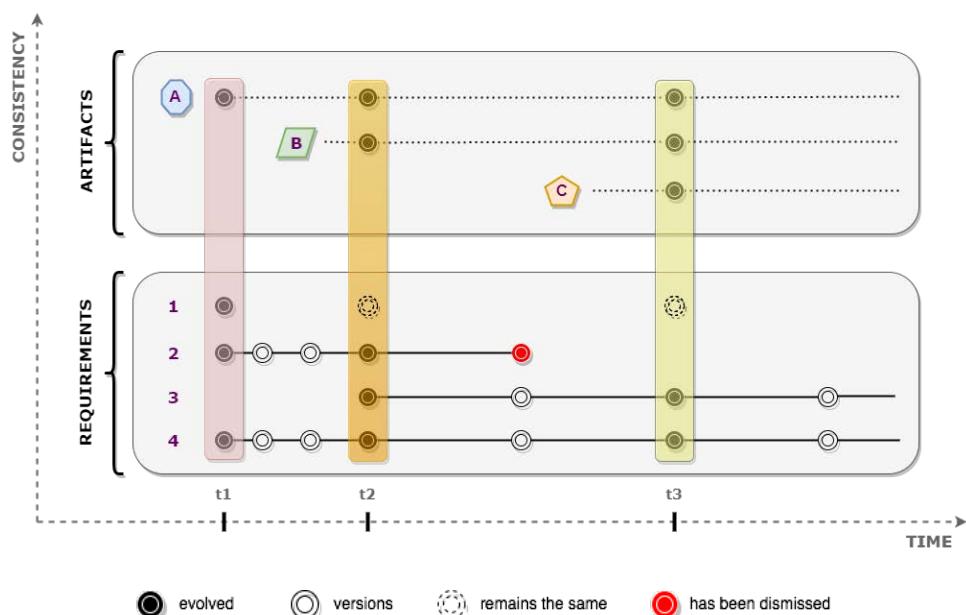


Figure 1. Requirements and artifacts being “photographed” in different phases of the project.

An example of such challenges is illustrated in Figure 1. Requirements and artifacts are supposed to evolve continuously along the project’s lifecycle. When looking at different moments of a project’s lifecycle, we should be able to guarantee that the set of existing requirements in a given time is consistent with the set of existing artifacts which models such requirements. For example, at the time “t1”, the project had 3 requirements (1, 2 and 4) and only 1 artifact concerned by them (A). So, at this time, only the artifact A should be consistent with requirements

1, 2 and 4. At the time “t2”, requirements 2 and 4 evolve and must be retested with respect to the artifacts. Besides that, a new requirement (3) came up and a new artifact (B) was designed, so now, it is both artifacts A and B that should be consistent with requirements 1, 2, 3, and 4. By pursuing the project, the requirement 2 was discontinued and a new artifact (C) was introduced, so at the time “t3”, artifacts A, B and C should be now consistent with requirements 1 (not yet evolved), 3 and 4, but not anymore with the requirement 2. That gives the dimension and the extent of challenges to consider when a large software system with multiple iterations is under development, not only because there are so many requirements and artifacts to align, but also because they come up, evolve, and are dismissed all the time along the project.

Another related challenge is that assessing interactive systems implies to assess system features with respect to the many possible data and system outputs that might occur when a user is interacting with them. This is an arduous testing activity due to the wide range of user tasks and the different combinations of testing data to assess. This problem is easier to be noticed in the testing of late artifacts such as full-fledged user interfaces when real test data are being manipulated. However, many other artifacts such as task models, and even preliminary versions of user interface prototypes can handle test data somehow. Verifying the consistency of supported test data in different artifacts is therefore another important source of testing.

In short, these concerns bring us three main challenges:

- formalize user requirements in such a way to provide testability in an ever-changing environment;
- guarantee consistency between user requirements and their representation in multiple artifacts; and
- lay on a flexible approach that could be reused to ensure such a consistency for newcomer artifacts along the project.

1.3. Objectives

The overall goal of this work is investigating methods for formalizing user requirements and automating the test of their functional aspects along the software development process. More specific goals include:

- Investigate a scenario-based approach aiming at specifying functional user requirements and their acceptance criteria in an understandable natural language for both technical and non-technical stakeholders.
- Allow the automated assessment of functional requirements on multiples user interface design artifacts.
- Provide assessment since early in the design process in order to ensure a full lifetime consistency between requirements and artifacts.
- Define a common-ground of concepts to specify user requirements and their acceptance criteria aiming at establishing a common vocabulary among the target artifacts.
- Implement a set of automated tools to support the approach and guarantee a high availability of tests throughout the development process of interactive systems.

Previous works have focused on modeling requirements and tests intrinsically coupled in a single artifact. The main supporting argument is that the specification of a requirement is only complete if it specifies the requirement’s acceptance criteria, i.e. under which conditions such a

requirement can be considered as done or accomplished. By doing this, requirements and tests could be kept updated more easily. In this context, Behavior-Driven Development (BDD) (Chelimsky *et al.*, 2010) has aroused interest from both academic and industrial communities as a method allowing specifying natural language user requirements and their tests in a single textual artifact. BDD benefits from a requirements specification based on User Stories (Cohn, 2004) which are easily understandable for both technical and non-technical stakeholders. In addition, User Stories allow specifying “executable requirements”, i.e. requirements that can be directly tested from their textual specification. By this means, they end up providing a “live” documentation once it contains, in a single artifact, the specification itself along with the automated testable scenarios which are able to certify whether some requirement has been attended or not. BDD also encompasses a scenario-based approach that benefit from an iterative cycle of producing-evaluating test scenarios in a final and implemented version of the system.

Despite its benefits providing automated testing of user requirements, BDD and other testing approaches focus essentially on assessing interactive artifacts that are produced late in the development process, such as full-fledged version of user interfaces. As far as early artifacts (such as task models, rough user interface prototypes, etc.) are a concern, such approaches offer no support for automated assessment. Besides that, the assessment of user requirements on user interfaces (and consequently on their related artifacts) requires the specification of the user’s interactive tasks that will be performed on such a UI. Despite defining a minimal template for specifying User Stories, BDD does not propose any other support to specify the user’s interactive tasks on such stories. While this freedom of writing gives to stakeholders a powerful approach for freely expressing their user requirements and interactive tasks, it requires that developers should implement each test scenario individually to allow them running on a fully implemented user interface and using low-level events that can hardly be reused to assess more abstract artifacts. In addition, it frequently gives rise to specifications of scenarios that, either do not encompass a description of interactive tasks, or do it but including several incompatible interactions such as clicks to be made on text fields in a form or selections to be made in a button.

To address these problems, we have studied the use of a formal ontology to act as common-ground for describing concepts used by platforms, models and artifacts that compose the design of interactive systems. The ontology was idealized to allow a wide description of interaction elements on user interfaces, as well as the behaviors associated with them. The aim of this ontology is therefore to support specification and testing activities in our approach by allowing that tests are written once, and then can be used to test all the set of considered artifacts. Whilst the ontology is aimed at being generic to many types of artifacts, in this thesis we have focused on its implementation for task models, prototypes and final user interfaces. As the automated assessment of these target artifacts was our guiding objective, after defining such an ontology, we designed the proposed approach already providing a fully support to the specification of consistent User Stories with an automated implementation ready for running the interactive behaviors recognized by the ontology directly on the target artifacts.

The ultimate goal of this thesis is therefore to present an approach based on BDD and User Stories to support the specification and the automated assessment of user requirements in software artifacts along the development process of interactive systems. The common-ground of concepts for describing the artifacts as well as the set of user-system interactive behaviors is provided by means of an ontology. By providing automated assessment of user requirements, we also target the guarantee of vertical traceability between them and the set of considered artifacts. As we aim to provide automated assessment since early in the design process, we have focused both on software artifacts describing early featuring aspects of the system, as well as on artifacts

implementing fully interaction aspects. We have limited the scope to user requirements describing functional aspects of the system, and to software artifacts aiming at describing the design of user interfaces (UIs). As such, we have focused on task models, user interface prototypes in different levels of refinement, and full-fledged (final) user interfaces as target artifacts.

1.4. Methodological Approach

Our approach aims at assessing user interface design artifacts from descriptions of interaction scenarios in User Stories, so two points are fundamental to demonstrate the validity of such an approach. The first one is the automatic verification of user requirements representation in our set of target artifacts. The second one is the translation of user requirements written by stakeholders into a language that allows the implementation of automated tests. To validate these two points, we have designed two case studies aiming at evaluating distinct aspects of our approach.

The first study was intended to investigate the level of understandability of User Stories specifications by a given group of stakeholders. To conduct this study, we have selected a group of potential Product Owners (POs) from the department in charge of business trips in our institute. POs are stakeholders that master the current business process and, in the case of this study, have the potential to eventually integrate a specialized group for specifying user requirements to maintain or develop a new software system in the business trip field.

During the study, the participants were invited, along structured interviews, to express a User Story they considered relevant within the group of system-related current tasks they work on daily. An important aspect we would like to evaluate was the spontaneous use of the interactive behaviors we had previously implemented in the ontology. With this objective in mind, we decided to present the BDD template for User Stories to the participants but omit the list of interactive behaviors we had modeled in the ontology. The stories produced were then evaluated for us in order to answer research questions related to:

- the level of understandability of User Stories structure by potential POs,
- identify in which extent predefined interactive behaviors presented in our ontology could be spontaneously used by potential POs, and
- the kind of adherence-to-the-template or adherence-to-the-ontology problems that would be identified in User Stories produced by potential POs.

With this exercise, we evaluated the set of User Stories produced by the participants and classified them according to their adherence to the User Story template initially presented, and to the predefined interactive behaviors modeled in the ontology. This analysis has been made separately for the first part of the User Story and for the related scenario, observing the existent gap between the steps each participant specified and the equivalent and available steps in the ontology. For each statement in the User Story, we have classified its adherence to the template or to the ontology in scales ranging from null adherence until full adherence. Additionally, we have categorized each deviation from the proposed template committed by the participants when writing their User Stories. They have been classified as adherence problems in categories such as lack of statement or keyword, understatement, misspecification, wrong information, minor writing complement, high-level of abstraction, and epic behavior. The complete experimental protocol as well as the results we got are presented in detail in chapter 7.

The second case study was intended to explore the translation of the stories produced by the participants into testable User Stories by using the set of predefined interactive behaviors as

proposed in the ontology. This study was also intended to demonstrate the potential of our approach to assess user interface design artifacts after having the User Stories formatted, besides identifying which kind of inconsistencies we would be able to point out by running our testing approach on such artifacts. As there is already a software system in production to book business trips in our institute (so we had no access to the software artifacts which were used to design such a system), we decided to apply *reverse engineering* (Chikofsky and Cross II, 1990) to obtain such artifacts from the software in production. We then redesigned the appropriate task models and user interface prototypes for the system.

To achieve the goals of this study, we conducted the following activities divided in 6 steps:

- Step 1: Format and add new User Stories based on the assets from the previous study and based on the current system implementation.
- Step 2: Add test cases to these User Stories.
- Step 3: Reengineer task models for the current system and run our approach to test the developed scenarios.
- Step 4: Reengineer user interface prototypes for the current system and run our approach to test the developed scenarios.
- Step 5: Run our approach to test the final user interface of the current system with the same developed scenarios.
- Step 6: Trace the results and verify the extent of inconsistencies we were able to identify in these multiple artifacts.

Finally, we analyzed the results of testing in each artifact by mapping such results to identify the trace of each inconsistency throughout the artifacts. That gave us a complete traceability overview of each step of the User Stories in the target artifacts. During the execution of each step of testing described above, we have collected and identified the reasons of failure in the mentioned artifacts in order to answer our research question concerning the kind of inconsistencies we are able to identify with this proposed approach. Such results allowed us to evaluate the effectiveness of the approach and to identify future improvement opportunities.

1.5. Thesis' Outline

This thesis is presented in nine chapters divided in four parts, as follows.

Part I – Introduction

The part I includes the present chapter and the chapter 2.

Chapter 2 Background

This chapter presents the state of the art about the concepts used in this thesis. It includes the main methods and techniques used for designing and modeling interactive systems following a scenario-based approach. It is presented a discussion about how Human-Computer Interaction (HCI) and Software Engineering (SE) communities handle the concept of User Stories and scenarios. As modeling activities by which our target artifacts are designed, a discussion about task modeling and user interface prototyping is also presented. These target artifacts will be explored respectively in chapters 5 and 6. Afterwards, we discuss the mechanisms for assessing the artifacts produced by such activities, focusing on the assessment of functional user requirements and GUI testing. We conclude this chapter with a discuss about software development processes and

methods that are typically used in SE for developing interactive systems, with an emphasis on Behavior-Driven Development (BDD).

Part II - Contribution

Chapter 3 **A Scenario-Based Approach for Multi-Artifact Testing**

This chapter is divided in three parts. The first one presents the rationale for the scenario-based approach we propose for specifying and testing user requirements on different artifacts. The second part presents the big picture of the micro-process that supports our approach to assess multi-artifacts, beginning with the proposed process being presented in a high-level view, with its activities packed and divided in production and quality assurance activities. Afterwards, an architectural view of the process is presented to point how the diverse software components and artifacts we consider are related for modeling requirements in a testable way. The chapter proceeds with the workflow view of the approach that presents how low-level activities are distributed for modeling and assessing such artifacts. The third and last part introduces the illustrative case study we base on for presenting the diverse stages of modeling and assessing artifacts. Therefore, this chapter addresses our specific goal of investigating a scenario-based approach aiming at specifying functional user requirements and their acceptance criteria in an understandable natural language for both technical and non-technical stakeholders.

Chapter 4 **Towards an Ontology for Supporting UI Automated Testing**

This chapter presents the ontological approach we have developed for specifying interactive behaviors and supporting our automated testing approach. The aim of the ontology described in this chapter is to support the assessment of user interface design artifacts as well as fully implemented user interfaces on interactive systems, providing a common and consistent description of elements that compose the semantics of interaction between users and systems in web and/or mobile environments. Therefore, this chapter addresses our specific goal of defining the common-ground of concepts to specify user requirements and their acceptance criteria aiming at establishing a common vocabulary among the target artifacts.

Chapter 5 **Modeling and Assessing Task Models**

This chapter details our strategy for modeling and assessing task models following our approach presented in chapter 3. The chapter begins by resuming the case study proposed in chapter 3, with task models being used to design user's tasks. By following this, we present firstly an orderly strategy for getting task models already consistent with the set of user requirements specified previously. In the second section, we explore our strategy for assessing the resultant task models. This section is presented in 3 steps. The first one refers to the extraction of possible scenarios from a designed task model, formatting them to meet the ontological pattern. The second one refers to the process of mapping elements from the task model for checking whether they are consistent with the respective elements in the User Stories, and hence with the ontology. Finally, the last step presents how our strategy has been implemented to support the testing in an automated way. Therefore, this chapter addresses our specific goals of providing early automated assessment of functional requirements on task models, with the support of automated tools to guarantee a high availability of tests throughout the development process of interactive systems.

Chapter 6

Modeling and Assessing User Interfaces: From Prototypes to Final UIs

This chapter details our strategy for modeling and assessing user interface prototypes following our approach presented in chapter 3. The chapter begins by resuming the case study proposed in chapter 3, with Balsamiq prototypes being used to design the user interface in a first stage of refinement. By following this, we present firstly how to produce UI prototypes already consistent with the set of user requirements specified previously. In the second section, we present how our previous developed ontology can support the development of prototyping tools able to produce consistent UI artifacts. The third section describes how we perform tests on fully implemented user interfaces by using an integrated multiplatform framework. This framework allows designing automated acceptance testing with low implementation efforts. The fourth section discuss how our approach supports the assessment of evolutionary UI prototypes, and how it could keep them consistent along the software development. Finally, the fifth and last section concludes the chapter pointing out advantages and limitations of this approach. Therefore, this chapter addresses our specific goals of providing early automated assessment of functional requirements on user interface prototypes in different levels of refinement, with the support of automated tools to guarantee a high availability of tests throughout the development process of interactive systems.

Part III - Evaluation

Chapter 7

Case Study 1 - Understandability of User Stories

This chapter presents the experimental design and the results of our first case study to evaluate the understandability of User Stories we used to model user requirements in our approach. To present our findings, this chapter is divided in 7 sections. The first one presents our experimental design, detailing our research questions and measures we used to assess the outcomes. Following this, we present the business narrative to give the context of how business travels are booked in our institute. Next, we detail our methodology to conduct the study, followed by the participant's profile, and the exercise we proposed to allow them writing their own User Stories. The sixth section brings the results of the study, highlighting the set of User Stories written by the participants, our adherence analyses considering stories and scenarios, our discussion of such results, our general findings and implications, and the threats to validity of this study. Finally, we conclude with our last remarks and point out future investigation opportunities in this field.

Chapter 8

Case Study II - Assessing User Interface Design Artifacts

This chapter describes the second case study we performed to evaluate our approach. The first section of this chapter presents the case study design, detailing how the study was planned and executed. The second section presents the set of complementary User Stories we have developed to support the design and testing of the artifacts developed for the case study. The third section adds a group of selected test cases with the aim of helping to validate such stories. The following sections present the modeling and testing results for each one of the assessed artifacts: task models, Balsamiq prototypes, and final UIs. In the seventh section, we build a traceability mapping to follow the inconsistencies found in each one of the target artifacts. Such mapping shows an edge-to-edge overall view of the testing scenarios, signalizing where a given step has failed in each artifact and why. We finish by presenting our findings and lessons learned, as well as our conclusions on the effectiveness of our testing approach, and the impact of the inconsistencies identified in the assessment of artifacts.

Part IV - Conclusion

Chapter 9 Conclusion

This chapter presents the final remarks about this thesis' work. We recapitulate our achievements and discuss the main contributions and limitations of the approach. We also provide some directions for future research in this field as well as our future works already planned to be conducted for improving the proposed approach. The chapter ends with the full list of publications resultant from this thesis.

Each chapter starts with a summary that presents the inner highlights. Moreover, whereas it is relevant, publications touching the core contributions of the chapter are presented at the end.

Chapter 2

Background

Summary

This chapter presents the state of the art about the concepts used in this thesis. It includes the main methods and techniques used for designing and modeling interactive systems following a scenario-based approach. The first part presents the methods for modeling user requirements for interactive systems including User Stories and Scenario-Based Design. At this part, it is presented a discussion about how Human-Computer Interaction (HCI) and Software Engineering (SE) communities handle the concept of User Stories and scenarios in a complementary perspective. These concepts are useful to understand how our approach related to previous works on scenario-based design and how these concepts are articulated to support our specification of User Stories.

Afterwards, the background about task analysis and modeling is presented along with a synthesis about how user interface prototyping contributes to the process of modeling interactive systems. An analysis about UI prototyping tools and how they have supported the modeling of user requirements over time is also presented. As task models and user interface prototypes constitute our target artifacts to be modeled and assessed by using our approach, such analysis is useful to present and align the concepts related to these artifacts.

The second part presents the mechanisms for evaluating user requirements, focusing on functional testing and GUI testing, which are target in our approach. We conclude this chapter with a contextualized discussion about software development processes that are typically used in SE for developing interactive systems. We explore the concepts of macro and micro processes that are used to define our approach and focus on the concepts related to agile methods and techniques, especially Behavior-Driven Development (BDD), on which our approach is based.

2.1. Methods for Modeling User Requirements for Interactive Systems

There are several methods for modeling user requirements. From traditional use cases until specific task models, user requirements modeling can assume different intents and abstraction levels. User-centered approaches usually model requirements using artifacts such as scenarios, task models and prototypes. In a scenario-based approach, these artifacts can be additionally aligned to provide a complete software design specification for interactive systems. Scenarios, however, have different meanings in the literature. They can also assume multiple forms and templates depending on the information requirements engineers want to highlight. In recent years, User Stories have stood out as one of the main scenario-based languages to specify automatable user requirements.

2.1.1. User Stories and Scenario-Based Design

Scenario-based design (SBD) is a family of techniques in which the use of a future system is concretely described at an early point in the development process. Narrative descriptions of envisioned usage episodes are then employed in a variety of ways to guide the development of the system. Like other user-centered approaches, scenario-based design changes the focus of

design work from defining system operations (i.e., functional specification) to describing how people will use a system to accomplish work tasks and other activities (Rosson and Carroll, 2001).

SBD follows an iterative design framework in which scenarios serves as a central representation of requirements throughout the development cycle, first describing the goals and concerns of current use, and then being successively transformed and refined through an iterative design and evaluation process (Figure 2). However, from analysis to evaluation, the SBD cycle does not tackle how to manage and assess the flow of artifacts that are produced all along these multiple development phases.

As central representation of requirements, scenarios can admit multiple templates according to the phase of development and to the level of abstraction that they are addressing for some information. Free narratives, for example, are useful in the very early phases, when typically, high-level business requirements are being defined (problem scenarios). Nevertheless, they are a frequent source of misunderstandings when used to refine requirements in activity or interaction scenarios in the design phase. Semi-formatted templates like in User Stories are better suitable in this case.

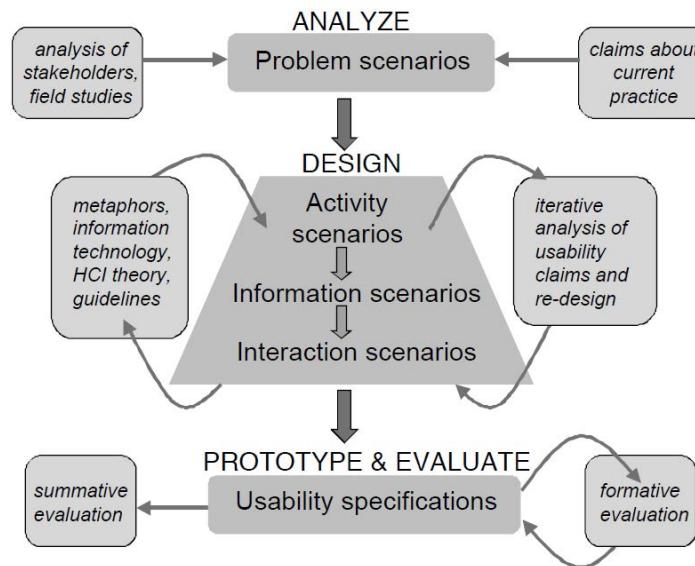


Figure 2. An overview of the scenario-based design (SBD) framework (Rosson and Carroll, 2002).

A large set of requirements can be expressed as stories told by users. Being a common activity in any requirements process, users and other stakeholders typically talk about their business process, emphasizing the flow of activities they need to accomplish. However, User Stories have a large meaning in the literature. The Human-Computer Interaction (HCI) community understands this concept as stories that users tell to describe their activities and jobs during typical requirements meetings. Being a common activity in any requirements process, users and other stakeholders typically talk about their business process emphasizing the flow of activities they need to accomplish. These stories are captured in requirements meetings and are the main input to formalize a requirements artifact. These meetings work mainly like brainstorm sessions and include ideally several stakeholders addressing different needs concerning features that may be developed. Iterative approaches capture these needs in successive meetings, according to the subject concerned in a particular iteration.

This concept of User Stories is close to the concept of scenarios given by Rosson & Carroll (Rosson and Carroll, 2001) and widely used in UCD design:

“Scenario spells out what a user would have to do and what he or she would see step-by-step in performing a task using a given system. The key distinction between a scenario and a task is that a scenario is design-specific, in that it shows how a task would be performed if you adopt a particular design, while the task itself is design-independent: it's something the user wants to do regardless of what design is chosen. Developing the scenarios forced us to get specific about our design, and it forced us to consider how the various features of the system would work together to accomplish real work.” (Lewis and Rieman, 1993).

According to Santoro (Santoro, 2005), scenarios are a well-known technique often used during the initial informal analysis phase. They provide informal descriptions of a specific use in a specific context of application, so a scenario might be viewed as an instance of a use case, representing a single path through it. A careful identification of meaningful scenarios allows designers to obtain a description of most of the activities that should be considered in a task model. Given task models have already been developed, Scenarios can also be extracted from them to provide executable and possible paths in the system.

In the Software Engineering (SE) side, User Stories are typically used to describe requirements in agile projects. This technique was proposed by Cohn (Cohn, 2004) and provides in the same artifact a narrative, briefly describing a feature in the business point of view, and a set of scenarios to give details about business rules and to be used as acceptance criteria, giving concrete examples about what should be tested to consider a given feature as done.

North (North, 2017) says that:

“A story should be the product of a conversation involving several people. A business analyst talks to a business stakeholder about the feature or requirement and helps them to frame it as a story narrative. Then a tester helps define the scope of the story – in the form of acceptance criteria – by determining which scenarios matter and which are less useful. A technical representative will provide a ballpark estimate of the amount of work involved in the story, and to propose alternative approaches. Many great ideas for systems come from the people developing them as well as the people who asked for them in the first place.”

Given requirements can emerge from multiple sources, including previous documentations, regulations, workflows, etc., after being captured, the User Stories need to be formatted, considering requirements emerged from other sources and looking for two main goals:

- (i) assure testability and non-ambiguous descriptions, and
- (ii) provide reuse of business scenarios.

For that, some formats and templates have been proposed (Wautelet et al., 2014). The most useful template however is given by Cohn and North (Cohn, 2004; North, 2017):

```
Title (one line describing the story)
```

Narrative:

```
As a [role]  
I want [feature]  
So that [benefit]
```

```
Acceptance Criteria: (presented as Scenarios)
```

Scenario 1: Title

```
Given [context]  
And [some more context]...  
When [event]  
Then [outcome]  
And [another outcome]...
```

```
Scenario 2: ...
```

This structure is largely used in Behavior-Driven Development (BDD) and has been named by North (North, 2017) as a “BDD story”. According to this template, a User Story is described with a title, a narrative and a set of scenarios representing acceptance criteria. The title provides a general description of the story, referring to a feature this story represents. The narrative describes the referred feature in terms of role that will benefit from the feature, the feature itself, and the benefit it will bring to the business. The acceptance criteria are defined through a set of scenarios, each one with a title and three main clauses: “Given” to provide the context in which the scenario will be actioned, “When” to describe events that will trigger the scenario and “Then” to present outcomes that might be checked to verify the proper behavior of the system. Each one of these clauses can include an “And” statement to provide multiple contexts, events and/or outcomes. Each statement in this representation is called step.

In the beginning of software development processes, requirements are more declarative and lead to User Stories in a high level of abstraction. As the project evolves, scenarios descriptions become more refined and closer to the user’s actions on the expected user interface. Chelimsky et al. (Chelimsky *et al.*, 2010) call them declarative and imperative scenarios. These two styles of writing tell the same stories, but at different levels of abstraction. It impacts different parts of the process in different ways. The first style is more horizontal, wrapping several activities up into a single step, which means it generally supports more scenarios, covering a larger set of features, but with fewer steps definitions. Conversely, the second style tends to be more vertical and customized to each scenario, with steps going step-by-step through performing each interaction on the user interface. It means that the work of writing steps spreads out more throughout the development, benefiting the development of test cases.

As we can realize, the approaches for scenarios from UCD and SE share the same concept. Both of them provide a step-by-step description of tasks being performed by users using a given system. The main difference between them lies in the testing and the business value components present in the SE approach. Scenarios from UCD, despite describing events that a given system can answer, do not describe the expected behavior from the system when those events are triggered, besides not determine the business motivation to develop the feature being described. Table 1 summarizes such characteristics.

Referring to what was said above in this section, we can conclude that to some extent, the approaches mentioned agree that User Stories and scenarios must provide a step-by-step description of tasks being performed by users using a given system. Nonetheless, there are some differences as illustrated by Table 1. This analysis gives us the opportunity to establish a correlation between requirements identified in User Stories, their representation in terms of tasks

and the extracted scenarios in both UCD and SE approaches. We can notice that the main difference lies in the degree of formality and their possible value to support automated tests. Another remark we can make is about the type of tasks mapped to scenarios in SE. As SE considers only tasks being performed by users when using an interactive system, User Stories in this context address only scenarios extracted from interactive tasks in task models. Cognitive tasks, for example, are not mapped to be SE scenarios because they cannot be performed in the system.

Approaches for User Stories and Scenarios	Key facts	Advantages	Shortcomings	Phases of development process	
				Early	Late
User Stories and/or scenarios by Rosson & Carroll (Rosson and Carroll, 2002)	Informal description of user activities contextualized in a story.	Highly flexible and easily comprehensive for non-technical stakeholders.	Very hard to formalize, little evolutionary and low reusability.	Yes	No
Scenarios extracted from task models by Santoro (Santoro, 2005)	A possible instance of execution for a given path in a task model.	Highly traceable for task models.	Dependency of task models and low testability.	Yes	No
User Stories and/or scenarios by North (North, 2017) and Cohn (Cohn, 2004)	Semi-formal description of user tasks being performed in an interactive system.	Highly testable and easily understandable for non-technical stakeholders.	Very descriptive and time consuming to produce.	Yes	Yes

Table 1. Approaches for describing User Stories and Scenarios.

This analysis gives us the opportunity to establish a correlation between requirements identified in User Stories, their representation in terms of tasks and the extracted scenarios in both UCD and SE approaches. A possible solution for a use case in the domain of air traffic control is presented in the Table 2 (adapted from (Santoro, 2005)).

Requirement	Task	Scenarios	
		Extracted from Task Models (UCD approach)	Written in the User Story template (SE approach)
Controllers should be able to select a plane in order to set its frequency Narrative As a controller I want to set frequencies for planes So that I can keep a private communication channel with them.	Select a plane (cognitive task)	First the controller identifies one of the planes not yet assumed	Given there are planes not yet assumed
	Click plane (interaction task)	Then the controller clicks on this plane to assume it	When I click on one of them
	Select a plane SC (cognitive task)	Then the controller decides to change the current frequency of one of the flights assumed	-
	Click FREQ (interaction task)	Then the controller clicks on the label FREQ to open the data-link menu	And I click on the label FREQ
	Open Menu (interaction task)	Then the controller opens the menu of frequency for this plane	And I open the menu of frequency for this plane
	Select Frequency (cognitive task)	Then the controller selects (in his/her head) a new frequency for this plane	-

	Click Frequency (interaction task)	Then the controller clicks on one of the available frequencies for this plane	And I click on one of the available frequencies
	Send (interaction task)	Then the controller clicks on the SEND button to send the new frequency to the aircraft	And I click on the SEND button
	-	-	Then the aircraft returns a double signal to confirm the new configuration

Table 2. Correlation between scenarios in UCD and SE approaches (adapted from (Santoro, 2005)).

Analyzing this correlation, we can note that the business value (represented in orange in the narrative) and the testing component (represented in green in the User Story scenario) allow us to implement test cases to validate the envisioned requirement, as well as checking when, after being implemented, this feature can be considered as done and correct (that correspond to the business value being achieved).

Another remark we can make it is about the type of tasks mapped to scenarios in SE. As SE consider only tasks being performed by users when using an interactive system, User Stories in this context address only scenarios extracted from interaction tasks in task models. Naturally, cognitive tasks, for example, are not mapped to be SE scenarios because they cannot be performed in the system.

2.1.2. Task Analysis and Modeling

Following an approach based on task models, interactive systems can be modeled to represent the flow of tasks that users should accomplish when using the system. According to Paternò (Paternò, 1999), tasks are activities that have to be performed to reach a goal. A goal is a desired modification of state or an attempt to receive state information. Each task is associated with one goal and each goal is associated with one or multiple tasks that can be represented in multiple abstraction levels.

2.1.2.1. Task Analysis

Task analysis is a process that aims to determine what the users do, the tools they use to do their work, the information they know or the information they must know for performing their work and is targeted to cover all or most cases and users. The general term task analysis can be applied to a variety of techniques for identifying and understanding the structure, the flow, and the attributes of tasks. Task analysis identifies the actions and cognitive processes required for a user to complete a task or achieve a particular goal.

According to the Usability BoK (*Usability Body of Knowledge*, 2018), a detailed task analysis can be conducted to understand the current system and the information flows within it. These information flows are important to the maintenance of the existing system and must be incorporated or substituted in any new system. Task analysis makes it possible to design and allocate tasks appropriately within the new system. The functions to be included within the system and the user interface can then be accurately specified. Some of the outputs of a task analysis include:

- a detailed description of physical, perceptual, and cognitive activities involved with each task,

- task duration and variability,
- task frequency,
- task sequence,
- task allocation,
- task complexity,
- environmental conditions,
- data and information dependencies,
- tools required for the task, and
- user skills, education, and training.

Cognitive Task Analysis (CTA) (Crandall, Klein and Hoffman, 2006) and Hierarchical Task Analysis (HTA) (Annett, 2003) are commonly used task analysis techniques. According to Hackos & Redish (Hackos and Redish, 1998), user and task analysis focuses on understanding:

- what users' goals are,
- what they are trying to achieve,
- what users actually do to achieve those goals,
- what personal, social, and cultural characteristics the users bring to the tasks,
- how users are influenced by their physical environment, and
- how users' previous knowledge and experience influence how they think about their work and the workflow they follow to perform their tasks.

A task analysis allows teams, for example, to discover what tasks a web site/app must support, determine the appropriate scope of content for an user interface, decide what applications your interface should include, refine or redefine the navigation or search for your website/app to better support users' goal, so to make sure the site is efficient, effective, and satisfying to users, build specific web pages and web applications that match users' goals, tasks, and steps, and ensure later on that the design supports all the tasks required. Additionally, the data for the task analysis can be assembled from several places including business requirements, user research, existing competitive products and brainstorming.

On the other hand, task analysis can be a very time-consuming activity if used with a high degree of detail on complex problems. It is possible to get caught in what is loosely termed "analysis paralysis" where more and more detail is investigated (Nicolle, 1999).

a. Procedure

Task decomposition: the aim of "high level task decomposition" is to decompose the high-level tasks and break them down into their constituent subtasks and operations. This will show an overall structure of the main user tasks. At a lower level it may be desirable to show the task flows, decision processes and even screen layouts.

The process of task decomposition is better represented as a structure chart (similar to that used in HTA). According to Dalkir (Dalkir, 2011), this shows the sequencing of activities by ordering them from left to right. In order to break down a task, the question should be asked "how this task is done?". If a sub-task is identified at a lower level, it is possible to build up the structure by asking "why is this done?". The task decomposition can be carried out using the following stages (Dalkir, 2011):

- i. Identify the task to be analyzed.

- ii. Break this down into between 4 and 8 subtasks. These subtasks should be specified in terms of objectives and, between them, should cover the whole area of interest.
- iii. Draw the subtasks as a layered diagram ensuring that it is complete.
- iv. Decide upon the level of detail into which to decompose. Making a conscious decision at this stage will ensure that all the subtask decompositions are treated consistently. It may be decided that the decomposition should continue until flows are more easily represented as a task flow diagram.
- v. Continue the decomposition process, ensuring that the decompositions and numbering are consistent. It is usually helpful to produce a written account as well as the decomposition diagram.
- vi. Present the analysis to someone else who has not been involved in the decomposition but who knows the tasks well enough to check for consistency.

Task flow diagrams: task flow analysis will document the details of specific tasks. It can include details of interactions between the user and the current system, or other individuals, and any problems related to them. Copies of screens from the current system may also be taken to provide details of interactive tasks. Task flows will not only show the specific details of current work processes but may also highlight areas where task processes are poorly understood, are carried out differently by different staff, or are inconsistent with the higher-level task structure (Dalkir, 2011).

Variations: if the tasks are already well understood, it may be sufficient to just identify and document the tasks as part of context of use analysis. According to Saffer (Saffer, 2006), the task analysis can consist in a raw list of features that the final application will have to carry.

2.1.2.2. Task Modeling

Task models provide a goal-oriented description of interactive systems but avoiding the need for the level of detail required for a full description of the user interface. Each task can be specified at various abstraction levels, describing an activity that has to be carried out to fulfil the user's goals. By modeling tasks, designers are able to describe activities in a fine granularity, for example, covering the temporal sequence of tasks to be carried out by the user or system, as well as any preconditions for each task (Paternò et al., 2017). The use of task models serves as multiple purposes such as better understanding the application under development (and in particular its use), being a "record" of multidisciplinary discussions between multiple stakeholders, helping the design, the usability evaluation, the performance evaluation, and the user in performing the tasks (acting as a contextual help). Task models are also useful as documentation of requirements both related with content and structure.

Task models rely on flexible and expressive notations providing systematic methods able to indicate how to use information in the task models. Each notation also provides automatic tools to model task information efficiently. Such notations represent task models by different syntaxes (both textual and graphical), different levels of formality, and different set of operators for task composition (Limbourg and Vanderdonckt, 2003). Hierarchical Task Analysis (HTA), GOMS, CTT and HAMSTERS are the most representative techniques notations for task modeling.

HTA (Annett, 2003) is a simple and flexible method that does not depend on a methodological context. It enables the representation of a task hierarchy that could be further detailed. Although HTA is task oriented and to some extent user oriented it still maintains a strong relationship with traditional software engineering. On the downside, there are no strict rules for creating an HTA diagram, so different analysts will generate inconsistent hierarchies at

varying levels of detail. HTA is not a predictive tool, it focuses on existing tasks and HTA diagrams can become quite complex. When used in large project, HTA requires a lot of overhead work/review and maintain task numbers and plans as tasks are edited and moved within the hierarchy. Also, it is difficult to synchronize the graphical and textual representations. The results of an HTA is a starting point for more detailed modeling methods, like GOMS. GOMS (Card, Newell and Moran, 1983) in its turn has some important limitations. It does not consider user errors or the possibility of interruptions. Only sequential tasks are considered. It can be inadequate for distributed applications (such as web-based applications).

ConcurTaskTrees (CTT) (Paternò, 2000) focus on activities and follows a hierarchical structure. It provides a graphical syntax with a rich set of temporal operators, besides task allocation, objects and task attributes. CTT can also be applied to multi-user applications, where users take on specific roles. The CTT notation is defined in terms of a hierarchical composition of temporal operators over named tasks, that relate a parent task to a non-empty set of child tasks. Tasks are associated with metadata including simple expressions over preconditions.

HAMSTERS (Martinie, Palanque and Winckler, 2011) is a task modeling language with a supporting tool. It is widely inspired by existing notations and tools and takes advantages from all of them. HAMSTERS has been implemented with the objective of making it easily extendable and it results in a CASE tool that contributes to the engineering of task models. HAMSTERS features a task model simulator as a dedicated API for simulating the execution of task sequences. It supports task types and temporal ordering, representation of information, knowledge, devices and objects (required to perform tasks), structuring mechanisms and collaborative activities.

Task modeling has a decisive impact on the design of UI prototypes once a dual-channel correspondence should be established between them, i.e. tasks described in task models must take place as an executable sequence of interactions on the user interface, and conversely, the user interfaces must support the execution of the whole set of possible interaction scenarios extracted from task models. As such, both artifacts must be kept in-line in order to guarantee the consistency between models, as well as the consistency between the models and the user requirements.

Ensuring the quality of models and their consistency with user requirements is an activity that demands manipulating such models in order to semantically compare their structure with a set of predefined requirements. Such manipulations result in a set of scenarios in which the model represents a valid interaction path in the system. This characteristic is particularly useful when identifying test scenarios for the system. Means of manipulating task models for obtaining test scenarios is a problem that has been recently studied by us (Silva and Winckler, 2017) and other authors (Bowen and Reeves, 2011; Campos *et al.*, 2017). Once a task model describes the whole set of tasks a user can perform in the system, besides providing the set of multiple paths that users are able to follow to accomplish such a task, test cases are obtained by going through these multiple paths, gathering a different execution scenario for each possible path. Therefore, all notations and tools for task modeling provide some kind of mechanism for extracting the set of possible scenarios by simulating a model execution.

In short, being scenarios informal descriptions of a specific use in a specific context, and task models, descriptions of possible activities and their relationships, scenarios support task development while task models can support scenarios identification.

2.1.3. User Interface Prototyping

A UI prototype is a previous representation of an interactive system. Prototypes are concrete artifacts and important components of the design process. They encourage communication, helping designers, engineers, managers, software developers, customers and users to discuss design options and interact with each other. They also permit early evaluation since they can be tested in various ways, including traditional usability studies and informal user feedback, throughout the design process (Beaudouin-Lafon and Mackay, 2000). Prototypes are often used in an iterative design process where the prototype is refined and become more and more close to the final user interface through the identification of user needs, constraints and feedbacks on early prototypes. It makes particularly important the investigation of multiple design options in the early phases. By running simulations on prototypes, we can determine potential scenarios that users can perform in the system.

Along this refining process, the prototype can be designed in different levels of fidelity. The prototype fidelity expresses the similarity between the final user interface (running in a particular technological space) and the prototyped UI. The UI prototype fidelity is said to be high if the prototype representation is the closest possible to the final UI, or almost in the same representation type. The fidelity is said to be low if the prototype representation only partially evokes the final UI without representing it in full details. Between high-fidelity and low-fidelity exists the medium-fidelity level, that gives more importance to the contents than the style with which these contents are presented (Coyette, Kieffer and Vanderdonckt, 2007).

Based on that, the design of user interfaces is expected to evolve along the whole software development process. While the beginning of the project requires a low-level of formality with UI prototypes being hand sketched in order to explore design solutions and clarify user requirements, the development phase requires more refined versions frequently describing presentation and dialog aspects of interaction. Full-fledged versions of user interfaces are generally produced only later in the design process, and frequently corresponds to how the user “see” the system. In the user’s point of view, the user interface actually *is* the system, so if some feature is not available there, then it does not exist. Such mature UI versions are also the source for acceptance testing and will be used by users to assert whether a system can be considered as done or not.

Prototyping is primarily a design activity in software engineering. It ensures that software prototypes evolve into technically sound working systems and serves for studying the effectiveness of particular designs. Several tools can help such an activity and many of them provide resources to evolve prototypes since sketching representations until the final design. Other features such as behavior specification, collaborative work, support for usability testing, etc. are also very useful for designers along the design process. In (Silva, Hak and Winckler, 2015; Silva *et al.*, 2017), we have evaluated a set of 104 commercial tools and 17 academic tools to investigate the availability of 13 essential features that emerged over time. Such features have been classified as milestones and encompass non-programming skills, pen-based interaction, widgets, behavior specification, collaborative work, reuse mechanism, scenario management, preview mode, support for usability testing, support for code generation, version control, annotations, and support for the entire design lifecycle.

From the 121 tools analyzed, we have noticed three milestones of releasing. The first period (before 1995) is characterized by the emergence of UIMS tools. UIMS tools focus on high-fidelity prototypes, using mostly design elements from the final interface, and being strongly dependent on the platform. UIMS tools lack the flexibility needed in the early phases of the development

process when designers should focus on problems to be solved in terms of business and users' requirements rather than terms of user interface design.

However, it is from this period the emergence of an important concept related to the design of user interfaces. The separation of components is a concept introduced by Green (Green, 1985) to separate the static and the dynamic aspects of an user interface (Figure 3). According to the author, the presentation component is responsible for the external presentation of the user interface, while the dialogue control component defines the structure of the dialogue between the user and the application program. Still according to the author, the dialogue control component can be viewed as the mediator between the user and the applications program. The user, through the presentation component, makes requests and supplies data to the application program. Unlike the presentation component, the dialogue control component must maintain a state and have control over it. The actions performed by this component will usually depend upon the context of the dialogue, therefore, any notations for it must be able to handle dialogue states and state changes. There is also the application interface model which is a component to define the semantics of the application. This representation includes the data objects that are maintained by the application, and the routines the user interface can use to communicate with the application. The concepts of presentation and dialog are part of our ontological definition of a user interface and will be explored in chapter 4.

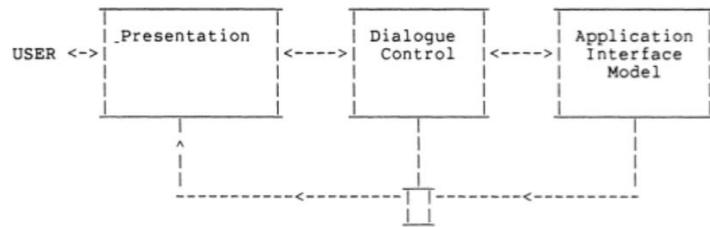


Figure 3. The logic model of a user interface (Green, 1985).

It is also from this period many reports of using tools such as PowerPoint and Visio to create user interface prototypes. Although PowerPoint and Visio are not intended to build prototypes, they provide functions for drawing presentations and creating transitions, which might have been helpful to build low-fidelity prototypes when no other UIMS tool was available.

The second identified period (1995-2005) encompassed tools with functionalities to support the development team when managing prototyping activities (ex. annotations, code generation, version control, etc.). There was an increasing interest in the period on alternative ways of prototyping user interfaces as well as in behavior modeling. For example, we observed the emergence of sketching tools such as SILK (Landay, 1996) and DENIM (Newman *et al.*, 2003).

The third and last period is characterized by a substantial increase of commercial tools and support for collaborative work. This period goes from 2007 to now. Along the three mentioned periods, features like Non-Programming Skills, the use of Widgets and Behavior Specification were the three most implemented by tools (over 70%). This fact can signalize the focus in providing a friendly environment for non-technical people since the first years. McDonald et al. (McDonald, Vandenberg and Smartt, 1988) in 1988 had already pointed the need to consider different skills from the various stakeholders involved and to allow them to use tools to design their own interfaces without technical skills. The way tools started providing that - and still remain until now - was through Widgets. Widgets have introduced a simple mechanism to encapsulate an idea

(and sometimes behaviors) for each component normally used to build GUIs. The concept of Widgets will be explored in chapter 6 when designing prototypes using the Balsamiq tool.

Features like Scenario Management, Support for Usability Testing and Support for the Entire Design Lifecycle are supported by a few tools (less than 10%). This number suggests a slow progress towards the support of the whole lifecycle of prototyping. Concerning Pen-Based Interaction, only 9.92% of tools implement this feature. Pen-Based Interaction feature was presented in SILK in 1995, and after some years, well-known tools like Adobe Illustrator and Photoshop implemented it. Nevertheless, it never seems to become a successful feature with commercial prototyping tools. This might be explained by the fact that sketches are hard to maintain (ex. ambiguity of sketches) and hard to make them evolve throughout the development process.

The five more covered milestones (Non-Programming Skills, the use of Widgets, Behavior Specification, Preview Mode and Reuse Mechanism) – all of them covered by more than half of tools – are also the oldest features presented by prototyping tools (since 1988). However, the availability of features like Behavior Specification, Preview Mode and Reuse Mechanism evolved along the time. Behavior Specification has benefited from more human-centered approaches such as Scenario-based specifications, while Preview Mode has incorporated co-execution between models and prototypes like in PetShop (Navarre, Palanque and Bastide, 2002) and ScreenArchitect. Since 2001, Reuse Mechanisms started to include technics like Plastic Interfaces (Calvary, Coutaz and Thevenin, 2001) and Responsive Design (Marcotte, 2014).

2.1.4. User Interfaces and Task-Based Development

Concerning the description of user interfaces, the Camaleon Framework (Calvary et al., 2002) treats the presentation and the dialog parts of an UI in three levels of abstractions: Abstract, Concrete and Final User Interfaces. The idea is that abstract user interface components (such as a Container) could be refined to a more concrete representation (such as a Window) that will ultimately feature a final implementation in a target platform (e.g. MacOs or Windows). User Interface (UI) specifications include more or less details according to the level of abstraction as shown in Figure 4.

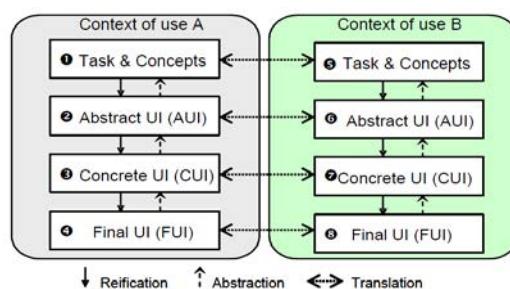


Figure 4. The Cameleon Reference Framework.

The UsiXML (USer Interface eXtensible Markup Language) (Limbourg *et al.*, 2004) implements the principles of the Cameleon framework in a XML-compliant markup language featuring many dialects for treating Character User Interfaces (CUIs), Graphical User Interfaces (GUIs), Auditory User Interfaces, and Multimodal User Interfaces. UsiXML is a declarative language that captures the essence of User Interface components. At a highest level of abstraction, UsiXML describes concepts of widgets, controls, containers, modalities and interaction techniques. UsiXML contain a few basic elements for describing the dialog part such as the concept of events, conditions and actions. For that, some authors have proposed to use a notation

based on statecharts called SWC (StateWebCharts) (Winckler and Palanque, 2003) to specify the UsiXML dialog. The same authors (Winckler *et al.*, 2008) have demonstrated that, using SWC, it is possible to describe the system behavior at different levels of abstraction using UsiXML.

As far as a common vocabulary is at a concern, the W3C published a glossary of recurrent terms for presentation components called MBUI (Model-based User Interface) (Pullmann, 2017). For the dialog component, SWC (Winckler and Palanque, 2003) and SXCML (State Chart XML: State Machine Notation for Control Abstraction) (Barnett, 2017) offer a language based on the state machine concepts.

There is also an intrinsically relationship between task modeling and user interface design. Some authors have even tried to establish a linguistic task modeling for designing user interfaces. Khaddam et al. (Khaddam, Mezhoudi and Vanderdonckt, 2015) presented a linguistic task model and notation. The model aims to separate the task and the semantic levels by adopting a well-defined set of task identification criteria. The provided notation enables identification of task input elements based on the task state diagram that is configured on each task. The notation also addressed the dynamic aspect of modeling by introducing dynamic tasks and pumping tasks.

Wolff et al. (Wolff *et al.*, 2005) proposes to link GUI specifications to abstract dialogue models. Specifications are linked to task models describing behavioral characteristics. Prototypes of interactive systems are refined and interactively generated using a GUI editor. The design cycle goes from task model to abstract user interfaces and finally to a concrete user interface. It is an interesting approach to have a mechanism to control changes in interface elements according to the task to which they are associated in the task models. However, the approach is not iterative and does not provide the necessary testing component to check and verify user interfaces against behavior-based user requirements.

Martinie et al. (Martinie *et al.*, 2015), followed by Campos et al. (Campos *et al.*, 2016), propose a tool-supported framework and a model-based testing approach to support linking task models to an existing, executable, and interactive application. The framework to define a systematic correspondence between the user interface elements and user tasks. The problem with this approach is that it only covers the interaction of task models with a concrete fully-functional user interfaces, not covering user interface prototypes or other types of requirements artifacts that can emerge along the process. Another problem is that it requires much intervention of developers to prepare the source code to support the integration, making it difficult to be adopted in applications that cannot receive interventions at the code level.

2.2. Methods for Evaluating User Requirements

Assuring the quality of user requirements representation is a complex task. Requirements can be expressed in so many forms and be represented through so many modeling and specification techniques that ensuring its consistency along the software development is a quite onerous task. Considering their representation as software artifacts, these last ones are usually only inspected manually in order to evaluate the adherence with other requirements representations. This process is part of what is called software verification. Software verification is defined as:

“(A) The process of evaluating a system or component to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase. (B) The process of providing objective evidence that the system, software, or hardware and its associated

products conform to requirements (e.g., for correctness, completeness, consistency, and accuracy)" (IEEE, 2017).

Boehm (Boehm, 1979) coined a quite famous question to simplify the definition of verification: "Am I building the product right?". This question aims to identify whether the software that is being built (or its intermediate outcomes, i.e. the software artifacts) actually meets the requirements, even if the software will not be exactly what the user is waiting for. By definition, verification involves the comparison between the requirements baseline and the successive refinements descending from it - the product design, detailed design, code, database, and documentation - in order to keep these refinements consistent with the requirements baseline (Boehm, 1979).

To certify that the software actually meets what the user is expecting, the software needs to be validated. Software validation is defined as:

"(A) The process of evaluating a system or component during or at the end of the development process to determine whether it satisfies specified requirements. (B) The process of providing evidence that the system, software, or hardware and its associated products satisfy requirements allocated to it at the end of each life cycle activity, solve the right problem (e.g., correctly model physical laws, implement business rules, and use the proper system assumptions), and satisfy intended use and user needs." (IEEE, 2017).

Boehm (Boehm, 1979) also coined an equivalent question to simplify the definition of validation: "Am I building the right product?". It means that validation identifies problems which must be resolved by a change of the requirements specification (Boehm, 1979). This is due to the fact that if a validation problem has been found, then the system actually does not satisfy the intended use and the user needs.

From the user point of view, evaluating his/her requirements usually means assessing a graphical user interface where he/she can effectively use the application and validate its behavior. This kind of validation made by final users is known as acceptance testing. Acceptance testing is a level of software testing where a system is tested for acceptability. The purpose of this test is to evaluate the system's compliance with the business and user requirements and assess whether it is acceptable for delivery (Graham *et al.*, 2008). By its nature, acceptance testing is usually focused on the functional aspect of the system.

2.2.1. Functional Testing

According to Myers (Myers, 2004), the purpose of software testing is to find errors so that they can be fixed. The term "errors" refers to any sort of problem with the system that could lead to a failure that could impact a user's experience. It is important to notice that "testing shows the presence, not the absence, of errors" (Dijkstra, 1970). It means that if a suite of tests is written, run, and discovers several errors, the tests have proven that those specific errors exist, and effort should now be expended to figure out how to fix them. This does not prove that those were the only errors in the system - in fact, it is impossible to write and/or run enough tests to prove that even simple functionality is absolutely correct (Dijkstra, 1970). Testing should be seen as an attempt to gain confidence that a system meets the expectations of its developers and users. Another important aspect of software testing is the reality that some errors do not matter if no one cares about them. There are far too many errors in any software system to fix them all, so

developers must always focus on those that are most likely to impact the user in a significant way (Hellmann, 2015).

In Software Engineering, the testing activity covers several levels of abstraction, from low-level testing such as unit and integration testing to high-level ones such as system and acceptance testing (Myers, 2004). The level of the artifact under testing determines the level of testing to be applied. This correspondence is shown in Figure 5. The V-model (Forsberg and Mooz, 1991) represents the multiple levels of testing according to the target artifact. The software source code, which is the lowest level of abstraction in terms of artifacts, is tested by unit testing. The software design and its architecture are tested by integration testing. At a higher level of abstraction, system requirements are tested by system testing while user requirements (that are being target in this thesis) are tested by acceptance testing. Low-level tests are aimed at assessing the quality of the code produced. As such kind of test is performed directly in the source code of the application, it is usually called “white box” testing. Contrarily, high-level tests are aimed at assessing the quality of the final product as a whole. As such kind of test is performed in the presentation layer of the application, it is usually called “black box” testing.

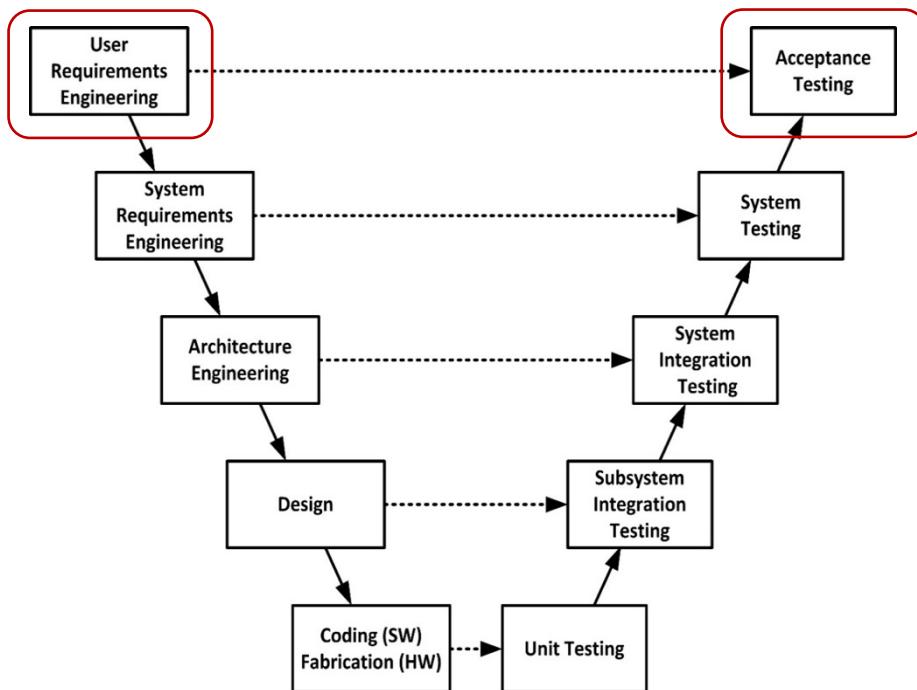


Figure 5. The V-model for testing.

Tests can also be focused on specific aspects of the system such as functionality, usability, scalability or performance. Among these several types of testing, we are focused on functional testing in this thesis. Functional testing aims to assess the functional aspect of user requirements. Functional testing identifies situations that should be tested to ensure the correct behavior of the system under development in accordance with the requirements previously specified. The acceptance testing refers to tests made under the client/user point of view to validate the right behavior of the system. For that, clients might be able to run their business workflows and to check if the system behaves in an appropriate manner.

Several techniques are employed to conduct functional testing such as Boundary Value Analysis, Equivalence Class Testing, Decision Table Base Testing, etc. (Myers, 2004). These techniques support the development of test cases that might be specified to validate the right

implementation of requirements. They explore the expected behavior of the system when performing the software features as well as the potential error situations that could lead to inconsistencies in the software behavior.

A big challenge related to testing software is that requirements are dispersed in multiple artifacts which describe such requirements in different levels of abstraction and in different perspectives according to the target audience. Thus, tests should run not only in the final product, but also in the whole set of artifacts to ensure that they represent the same information in a non-ambiguous way, and in accordance with the whole requirements chain. Moreover, testing should be performed along the development process as clients and users introduce new demands or modify the existing ones all along the iterations. Regression testing is then crucial to ensure that the system remains behaving properly and in accordance with the new requirements introduced. Manual regression testing however is extremely time consuming and highly error-prone. Therefore, automated testing is a key factor to support testing in an ever-changing environment, allowing a reliable checking of requirements and promoting a high availability of testing.

2.2.2. GUI Testing

Being the main bridge between the system and the end user, graphical user interfaces (GUIs) are a crucial target artifact for testing. As Hellmann (Hellmann, 2015) pointed out, the simplest way to perform GUI testing is with manual testing, wherein a human tester interacts with an application to verify that its responses are correct. A human tester can easily interact with an application and recognize when an error occurs, but manual testing is very slow and error-prone. If testing should be done frequently, then manually testing a GUI quickly becomes unfeasible. Automated tools exist to simplify and automate this process. Most GUI testing tools work on the capture/replay paradigm. In capture/replay, testing tools monitor the set of interactions between a human tester and the system and record these steps so that they can be replayed later as automated tests. However, capture/replay tools (CRTs) do not tend to record tests in a human-readable manner, meaning that it is much more difficult to modify an existing test than to record a new one.

Other tools are designed to make direct calls to the system using the native support for automation of each user interface environment. When testing user interfaces presented by means of a web browser, for example, such tools make calls directly to the browser. How these direct calls are made, and the features they support depends on the target browser. Such approaches tend to be much more flexible to implement automated testing for GUIs. Tests specified by these approaches tend to be easier to maintain, but they carry the same problem of low human-readability. Some tools have then emerged to raise the level of automated test specification. With tools like JBehave¹, users can specify and run their own text-based User Stories to automate acceptance testing, which allows “out-in” development, i.e. end-users being empowered to guide the software development by writing their own automated user requirements and tests.

2.2.3. Artifacts Inspection and Requirements Traceability

Artifacts other than user interfaces are not commonly tested. A common argument is that they cannot be “executed” in order to be tested. The set of user requirements they represent is usually only inspected manually in a try to verify its consistency. Inspections can be of different types including formal technical reviews, walkthroughs, peer desk check, informal ad-hoc feedback, and so on (Tian, 2005). On another front, requirements traceability techniques have been studied

¹ <http://jbehave.org>

for a long time as a way to trace such requirements along their multiple versions (horizontal traceability) or along their representation in another artifacts (vertical traceability) (Ebert, 2011).

Some authors concentrated efforts in providing automated tools to keep compatibility between different artifacts models. Those approaches, regardless providing some mechanism to trace or assess requirements for particular environments, do not consider how to integrate and test the set of multiple other artifacts that are commonly used throughout development processes. Luna et al. (Luna *et al.*, 2010), for example, propose WebSpec, a requirement artifact used to capture navigation, interaction and UI features in web applications, where diagrams can be validated due to the automatic derivation of interaction tests. WebSpec can be used in conjunction with mockups to provide realistic UI simulations, allowing quick requirements validation. It can also be used to capture requirement changes and use them to semi-automatically upgrade the application and maintain quality standards.

Buchmann and Karagiannis (Buchmann and Karagiannis, 2017) presented a modeling method for the elicitation of requirements for mobile apps that enables semantic traceability for the requirements representation. Instead of having requirements represented as natural language items that are documented by diagrammatic models, the communication channels are switched: semantically interlinked conceptual models become the requirements representation, while free text can be used for requirements annotations/metadata. The authors claim that the method can support semantic traceability in scenarios of human-based requirements validation but using an extremely heavy modeling approach which is not suitable for checking requirements in a high level of abstraction. Besides that, the method is not focused on providing a testing mechanism through common artifacts, but only in validating the requirements modeled within the approach.

2.3. Software Development Processes

Since the software crisis in 1968, software development processes have emerged as a silver bullet for delivering high-quality, scalable and reliable software systems. The waterfall model (Royce, 1970) guided many software development processes over time proposing a seven-step cascading model to produce software covering activities from system requirements until operation. This model implements the concept of Big Design Up Front (BDUF) where the software design phase is fully completed before the implementation is started.

The waterfall model gave rise to several development problems such as the difficulty of designers to foresee problem areas without extensive prototyping and at least some investment into implementation, the difficulty of evolving requirements once clients may not know exactly what their requirements are before they see working software, the taking of bad design decisions due to the lack of knowledge about the system at the beginning of the project, etc. Contrary to this model, iterative processes have emerged as a solution to break the software process in small iterations into continuous cycles of development. Whilst the waterfall model delivers a big software outcome only at the end of the process, iterative processes give more flexibility focusing on delivering smaller, but incremental software deliverables along multiple iterations. An illustration of both waterfall and iterative models is presented in Figure 6.

Both waterfall and iterative models served as basis for many software development processes since then. Such processes are said to be macro-processes. Macro-processes emphasize the overall external behaviors of processes, whilst micro-processes emphasize the internal workings of processes (Osterweil, 2005).

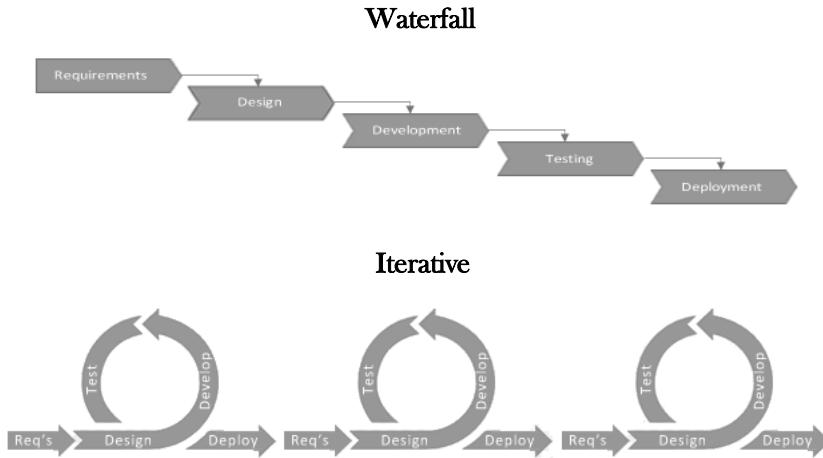


Figure 6. Simplified versions of waterfall and iterative models.

Macro-processes are concerned by the overall software development lifecycle, i.e. the choice of models (waterfall, iterative, etc.) to conduct the project's lifecycle affects the macro-process. Over time, based on their characteristics and emphasis, macro-processes were being classified as traditional and agile methodologies. The Unified Process (UP) (Jacobson, Booch and Rumbaugh, 1999) became the most known example of traditional method. UP is a heavyweight multi-phase software development process that emphasizes proven design, extensive documentation, and detailed planning. UP is considered as an architecture-centric, use-case driven and risk-focused process framework which benefits from a well-structured object-oriented modeling language, the Unified Modeling Language (UML) (Booch, Rumbaugh and Jacobson, 2005). Opposite to traditional methods, agile methods consist in a lightweight set of methods focused on communication between users and developers, short-time software delivery, adaptive planning, and self-organization. Agile methods are adaptive rather than predictive, and people-oriented rather than process-oriented. Such methods are detailed hereafter in the next subsection.

Differently from macro-processes, micro-processes are concerned by the analysis and design techniques, i.e. the set of techniques chosen to be used within the different phases of the project's lifecycle affects the micro-process instead. The approach we propose in this thesis therefore defines a micro-process once it affects the strategies for specifying and assessing user requirements on multiple software artifacts within the different phases of the project. The micro-process designed for our approach is presented in chapter 3.

2.3.1. Agile Methods

During the 1990's, a number of lightweight software development methods evolved in reaction to the prevailing heavyweight methods. They became known in early 2001 as agile methods. Such methods advocate for an incremental and iterative paradigm with a Rough Design Up Front (RDUF) (Ambler, 2002). Agile methods follow four main values expressed in the Manifesto for Agile Software Development (Beck et al., 2001). Such values emphasize:

- **individuals and interactions** over processes and tools,
- **working software** over comprehensive documentation,
- **customer collaboration** over contract negotiation, and
- **responding to change** over following a plan.

Agile methods are based on sustainable development, collaboration between business people and developers, self-organizing teams, working software adding business value, continuous delivery of software, changing requirements, short development timescales, motivated individuals, face-to-face conversation, technical excellence and emergent design, simplicity, working software as measure of progress, and continuous evaluation. These principles are declared in the manifesto as follows:

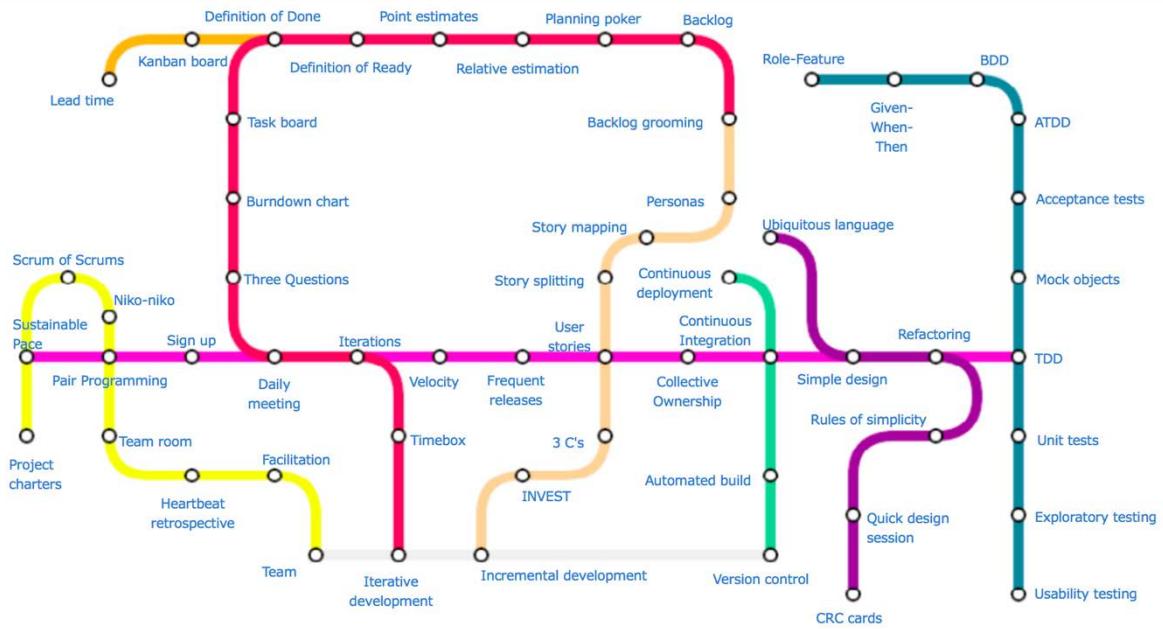
- Our highest priority is to satisfy the customer through early and continuous delivery of valuable software.
- Welcome changing requirements, even late in development. Agile processes harness change for the customer's competitive advantage.
- Deliver working software frequently, from a couple of weeks to a couple of months, with a preference to the shorter timescale.
- Business people and developers must work together daily throughout the project.
- Build projects around motivated individuals. Give them the environment and support they need and trust them to get the job done.
- The most efficient and effective method of conveying information to and within a development team is face-to-face conversation.
- Working software is the primary measure of progress.
- Agile processes promote sustainable development. The sponsors, developers, and users should be able to maintain a constant pace indefinitely.
- Continuous attention to technical excellence and good design enhances agility.
- Simplicity - the art of maximizing the amount of work not done - is essential.
- The best architectures, requirements, and designs emerge from self-organizing teams.
- At regular intervals, the team reflects on how to become more effective, then tunes and adjusts its behavior accordingly.

This list of twelve principles are implemented through a series of practices that vary from method to method. The “subway map” to the agile practices implemented by leading approaches is illustrated in Figure 8. Popular agile software development methods include (but are not limited to): Adaptive Software Development (ASD) (Highsmith, 1999), Agile Unified Process (AUP) (Ambler, 2005), Disciplined Agile Delivery (DAD) (Ambler and Lines, 2012), Dynamic Systems Development Method (DSDM) (Stapleton and Constable, 1997), Extreme Programming (XP) (Beck and Andres, 2004), Feature-Driven Development (FDD) (Palmer and Felsing, 2002), Lean Software Development (Poppendieck, Poppendieck and Poppendieck, 2003), Kanban (Anderson, 2010), Rapid Application Development (RAD) (Martin, 1991), Scrum (Schwaber, 2004), and Scrumban (Ladas, 2009; Reddy, 2015).

The need of modeling is quite controversial in agile methods. They consider in general that, to be effective, agile modelers should know a wide variety of modeling techniques so that they have the skills and knowledge to apply the right artifact(s) for the situation at hand (Ambler, 2002). Modeling practices always include a strong commitment with executable specifications, modeling only the essential at a given time (keeping the RDUF paradigm), and an automated test-first approach. Figure 7 illustrates such commitments according to Ambler (Ambler, 2002).

UI prototypes are also an important modeling artifact explored by agile methods. Käpyaho and Kauppinen (Käpyaho and Kauppinen, 2015) have investigated how prototyping can solve the challenges of requirements in an agile context. The authors suggest that prototyping can solve some problems of agile development, such as the lack of documentation, poor communication tools, but it also needs complementary practices such as the use of ATDD. The authors conclude

that one of the biggest benefits from prototyping is that prototypes act as tangible plans that can be relied on when discussing changes.



Lines represent practices from the various Agile "tribes" or areas of concern:



Figure 8. “Subway Map” to agile practices (*Agile Alliance*, 2018).

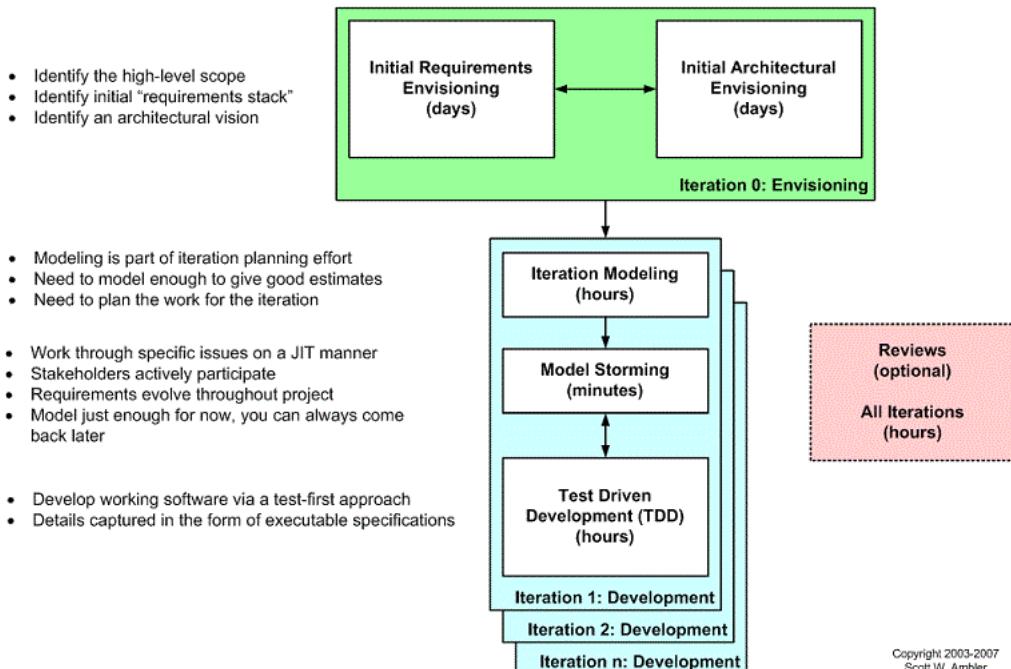


Figure 7. Agile Model Driven Development (AMDD) (Ambler, 2002).

2.3.2. Behavior-Driven Development

Behavior-Driven Development (BDD) (Chelimsky *et al.*, 2010) is an evolution of Test-Driven Development (TDD) (Beck, 2002; Astels, 2003), and is intended to make the practice of writing automated testing more accessible and intuitive to newcomers and experts alike. It shifts the vocabulary from being test-based to behavior-based. It positions itself as a development paradigm, emphasizing communication and automation as equal goals. In BDD, the behaviors represent both the requirements specification and the test cases. According to North (North, 2009):

“BDD is a second-generation, outside-in, pull-based, multiple-stakeholder, multiple-scale, high-automation, agile methodology. It describes a cycle of interactions with well-defined outputs, resulting in the delivery of working, tested software that matters.”

BDD has aroused interest from both academic and industrial communities in the last years. Supported by a wide development philosophy that includes Acceptance Test-Driven Development (ATDD) (Pugh, 2010) and Specification by Example (Adzic, 2011), BDD drives development teams to a requirements specification based on User Stories (Cohn, 2004) in a understandable natural language format. This format allows specifying executable requirements by means of a Domain-Specific Language (DSL) provided by Gherkin². Gherkin is a business readable DSL that lets users and developers describe software's behavior without detailing how that behavior is implemented. Gherkin serves two purposes: documentation and automated tests. By using this language, requirements specifications can directly be used to implement automated tests, conducting to a “live” documentation and making easier for clients and other stakeholders to set their final acceptance tests. It guides the system development and brings the opportunity to test scenarios directly on the user interface with the aid of testing frameworks for different platforms.

In BDD, the user's point of view about the system is captured by the User Stories, described according to a template. The BDD approach assumes that clients and teams can communicate using this semi-structured natural language description, in a non-ambiguous way (because it is supported by test cases). These test cases are developed for each unit of software feature following a TDD approach which encompasses:

- define a test set for the unit first,
- make the tests fail,
- then implement the unit,
- finally verify that the implementation of the unit makes the tests succeed.

BDD specifies that tests of any unit of software should be specified in terms of the desired behavior of the unit (North, 2006), i.e. the behavior that adds business value to the product. Such behaviors are specified in the User Stories. Additionally, BDD extends the TDD philosophy by (*Agile Alliance*, 2018):

- Applying the “Five Why's” principle to each proposed User Story, so that its purpose is clearly related to business outcomes,
- Thinking “from the outside in”, in other words implement only those behaviors which contribute most directly to these business outcomes, so as to minimize waste,

² <https://github.com/cucumber/cucumber/wiki/Gherkin>

- Describing behaviors in a single notation which is directly accessible to domain experts, testers and developers, so as to improve communication,
- Applying these techniques all the way down to the lowest levels of abstraction of the software, paying particular attention to the distribution of behavior, so that evolution remains cheap.

BDD is the primary software development method for specifying automated natural language user requirements. Efforts to specify requirements in a natural language are not recent though. Language Extended Lexicon (LEL) (Leite and Oliveira, 1995) have been studied since the 90's. The authors propose a lexical analysis of requirements descriptions in order to integrate scenarios into a requirements baseline, making possible tracing their evolution. They were followed by other attempts to identify test cases from requirements specified in natural language (Sneed, 2007; Dwarakanath and Sengupta, 2012).

BDD has been evaluated (Lopes, 2012) and its characteristics analyzed and studied (Solís and Wang, 2011; Egbreghs, 2017) by several authors. Studies have been conducted to explore the use of BDD as part of empirical analysis of acceptance test-driven development (Melnik, 2007), to support enterprise modeling within an agile approach (Valente *et al.*, 2017) and within an user-centered approach (Valente *et al.*, 2016), to support requirements engineering with gamification (Lombriser *et al.*, 2016), to support a testing architecture for micro services (Rahman and Gao, 2015), to support the analysis of requirements communication (Oran *et al.*, 2017), to support safety analysis and verification in agile development (Wang and Wagner, 2018), and to enhance the critical quality of security functional requirements (Lai, Leu and Chu, 2014). Other studies have concentrated in the use of automated acceptance testing to support BDD traceability (Lucassen *et al.*, 2017), or in analyzing its compatibility with business modeling (Carvalho, Carvalho e Silva and Manhaes, 2010; Carvalho, Manhães and Carvalho e Silva, 2010) and with BPMN (Lübke and Van Lessen, 2016).

BDD has also been used to support implementation of source code. Soeken *et al.* (Soeken, Wille and Drechsler, 2012) propose a design flow where the designer enters in a dialog with the computer where a program processes, sentence by sentence, all the requirements creating code blocks such as classes, attributes, and operations in a BDD template. The template proposed by the computer can be revised; which leads to a training of the computer program and a better understanding of following sentences.

2.4. Conclusion

The background presented in this chapter points towards a gap when integrating different requirements artifacts throughout a design process. Some methods addressed concerns in scenarios descriptions, other ones in UI prototyping or task modeling, but none of them addressed the problem of integrating the assessment of multiple artifacts in order to ensure correctness and consistency of user requirements modeling along the software development. In the next chapter, we start to present our approach to address such mentioned gaps, first identifying a scenario-based approach aiming at addressing the concerns of specification, followed by a micro-process for implementing such an approach.

2.5. Resultant Publications

Silva, T. R., Hak, J.-L., Winckler, M. & Nicolas, O. (2017). A Comparative Study of Milestones for Featuring GUI Prototyping Tools. *Journal of Software Engineering and Applications*, 10 (06), pp. 564-589. DOI: <http://doi.org/10.4236/jsea.2017.106031>. (Silva *et al.*, 2017)

Silva, T. R., Hak, J. L. & Winckler, M. (2015). A Review of Milestones in the History of GUI Prototyping Tools. In: INTERACT 2015 Adjunct Proceedings: 15th IFIP TC. 13 International Conference on Human-Computer Interaction, pp. 267-279, vol. 22. University of Bamberg Press. (Silva, Hak and Winckler, 2015)

Part II - Contribution

Chapter 3

A Scenario-Based Approach for Multi-Artifact Testing

Summary

This chapter presents an approach based on scenarios to support the specification and the multi-artifact assessment of functional user requirements along the development process of interactive systems. The approach is aimed at describing how the different artifacts, inputs and outcomes should be used to support activities of specification and assessment of requirements. The approach relies on the premise that user requirements, commonly expressed by the means of User Stories and scenarios, must be specified in a certain way to be employed in automated testing of the various artifacts used along the development process. For that purpose, the approach employs a user interface ontology that ensures that elements described in the scenarios refer to elements described in the artifacts. The focus of this chapter is to present a big picture of the approach and its inner rationale. The ontology itself is presented in chapter 4, while the instantiation of the approach to specific artifacts is presented latter on in chapters 5 (for task models) and 6 (for user interfaces).

In the present chapter, the approach is presented in 3 different views. The first one is a high-level view with its activities packed and divided in Production and Quality Assurance activities. We show in this view how different roles contribute to the approach by writing testable User Stories. Afterwards, an architectural view of the approach is presented to point how the diverse software components and artifacts are related for modeling requirements in a testable way. This view is complemented by a workflow view of the approach that presents how low-level activities are distributed for modeling and assessing user interface design artifacts. The workflow view addresses responsibilities for the multiple roles involved in the process as well as the resources that should be produced or delivered throughout the activities flow. Activities in the workflow are presented through a set of steps that could be followed by stakeholders for modeling requirements in a behavior-oriented way, allowing them to be properly tested afterwards. We also discuss alternatives for performing the approach depending on the stage of the project in which the approach is employed.

Lastly, an illustrative case study for assessing a generic web system for booking flight tickets is presented to guide and exemplify the use of the approach. The study is organized following the set of step-by-step activities proposed by our workflow. The same case study is retaken to provide consistent examples of use throughout the following chapters.

3.1. Rationale for a Scenario-Based Approach

Requirements are the main source of information for specifying a software system, but they are not necessarily explicit or formally specified. They can emerge from multiple sources. In addition to requirements expressed by the stakeholders, requirements might have origin in documents such as business models, laws and regulations. As such, several aspects of information, from the macro business goals until the most detailed information about user tasks are modeled

in several requirements artifacts, designing different aspects of the system (e.g. business models, use cases, task models, etc.). As many stakeholders have different views of the system and different phases of development require distinct information, artifacts used for modeling tend to be very diverse throughout the development, ranging from business models in the very beginning of the project, until complete test specifications at the end. In iterative processes, the cycle of producing and evaluating requirements and artifacts permeates all phases of system development, from requirements and business analysis until the software testing (Jacobson, Booch and Rumbaugh, 1999).

When designing new software systems, clients and users are keen to introduce new requirements along successive iterations and such requirements tend to vary widely, once different stakeholders bring different requirements to the product. Clients are typically involved in bringing requirements that limit the budget, the scope and the timeframe available for development. Requirements are cut and/or introduced based on such requirements. Business people bring typically high-level and macro requirements that drive the project to a business goal to be achieved, while users are aimed to set more functional requirements that specify practical features the software should provide.

Such characteristic has an impact in the forthcoming development as well as in previously developed artifacts. Given requirements should be verified and tested against not only the software already produced, but also against the other permanent artifacts produced throughout the process (Boehm, 1979), it leads us to a cycle of permanent production of multiple artifacts, in multiple versions, evolving all along of multiple phases of development until they reach the status of final product. Traces along those multiple evolutions should be maintained for quality assurance purposes (Ebert, 2011). This cycle is illustrated in Figure 9.

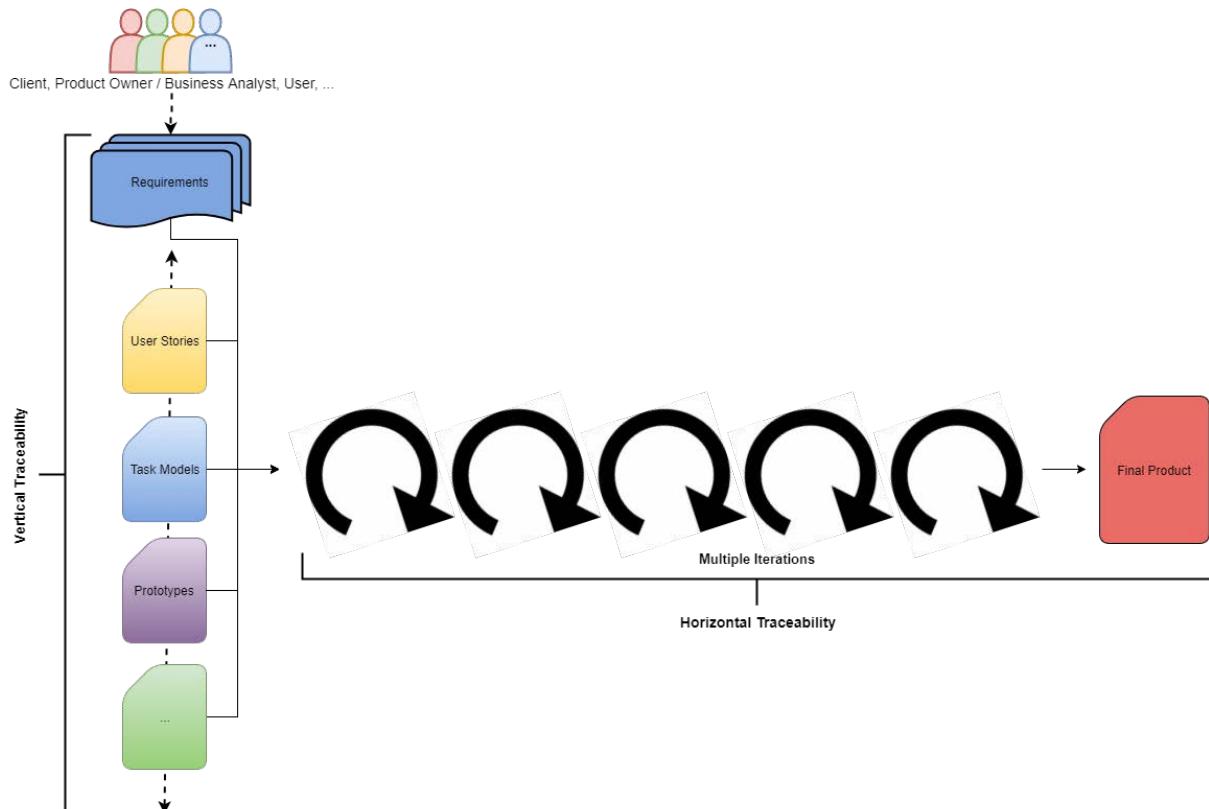


Figure 9. The cycle of permanent evolution of artifacts in iterative processes

Among the several types of requirements that can emerge during a specification, we are especially interested in the ones which model business and functional aspects. Business requirements relate to a business' objectives, vision and goals. They also provide the scope of a business need or problem that needs to be addressed through a specific activity or project. Functional requirements break down the steps needed to meet the business requirements. When developing functional requirements, a comprehensive list of steps that will be taken during the project is developed. Functional requirements are very detailed and provide information on how business needs and goals will be delivered through a specific project.

Previous works have suggested that Use Cases (Bertolino *et al.*, 2006) can be used to specify functional requirements and extract scenarios to be tested against the system. Nonetheless, scenarios can be extracted and/or formalized from information available in many artifacts such as business models (Carvalho, Carvalho e Silva and Manhaes, 2010; Carvalho, Manhães and Carvalho e Silva, 2010), task models (Paternò and Mancini, 1999), and prototypes (Elkoutbi, Khriss and Keller, 2006). Based on that, we suggest that scenarios can be a suitable alternative to start analyzing the relationship between functional requirements expressed using diverse artifacts.

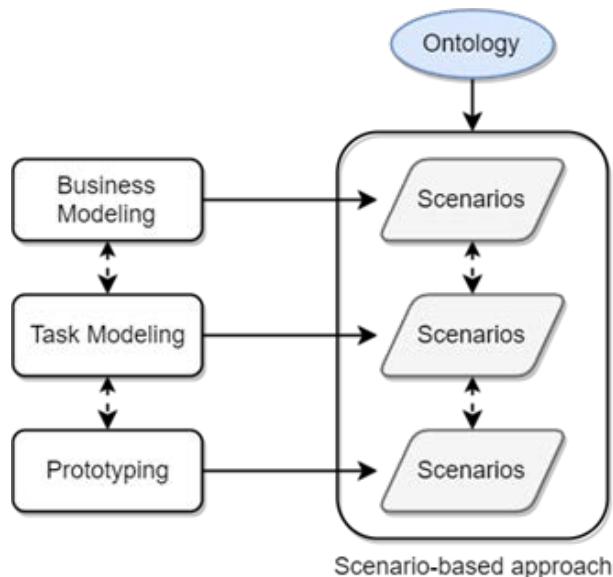


Figure 10. Modeling business and functional requirements in a scenario-based approach.

Therefore, for modeling business and functional requirements, we propose a scenario-based approach, taking multiple views of the system into account. Figure 10 illustrates this approach, so far designed to support three modeling processes: business modeling, task modeling and prototyping. The processes of business and task modeling as well as the process of prototyping are iterative and contribute mutually for the development of each one. The relationship between task modeling and prototyping are quite natural once both compose the typical process of modeling user requirements for interactive systems. Both of them are also innately scenario-based as they use scenarios to perform and simulate user activities in the system.

The relationship between business and task models has already been studied by some authors (Pontico, Farenc and Winckler, 2007; Sousa, Mendonça and Vanderdonckt, 2008; Winckler and Palanque, 2012). Winckler and Palanque (Winckler and Palanque, 2012) have demonstrated how – starting from a business process – task models can be designed to specify the flow of detailed tasks that a user should accomplish to perform a given activity for each business process.

With this perspective, the process of business modeling can also fit in a scenario-based approach, once the overall business view about the system can be easily described using a scenario narrative.

Artifacts produced by the activities of task modeling and prototyping have been chosen as target artifacts because they compose what we call user interface (UI) design artifacts, i.e. artifacts typically used to design and support the development of UIs for interactive systems. As these artifacts are essentially different in their conception, usefulness on different stages of development, and nature of information modeled, the strategy for testing them is supposed to vary for each one, or at least for groups of them. We classified such artifacts in two groups. The first one encompasses artifacts typically used in the early stages of development for modeling aspects of interaction and/or navigation. We have classified task models and UI prototypes with a low level of refinement in this group. The second group encompasses artifacts typically used later in the development process for designing more detailed (or even definitive) aspects of interaction and navigation. We have classified iterative UI prototypes as well as final UIs in this group.

The problem raised when using the aforementioned artifacts is that there is not a standard method to specify scenarios for them. They can be freely described following few or no templates, from informal descriptions such as textual narratives until more formal ones such as pre-formatted lists of tasks extracted from task models. It makes very hard the work of identifying similar requirements that eventually describe the same features but in different perspectives. To tackle this problem, we explored the use of an ontological support aiming at describing common behaviors with a standard vocabulary for writing User Stories as scenario artifacts. The main benefit of this strategy is that User Stories described following a common vocabulary can be directly automated for running test scenarios on other artifacts. As the common vocabulary has been set using well-established concepts such as UsiXML (Limbourg *et al.*, 2004), W3C MBUI (Paternò *et al.*, 2017) and others, the resultant ontology establishes indeed the searched common ground for a scenario-based approach considering multiple artifacts.

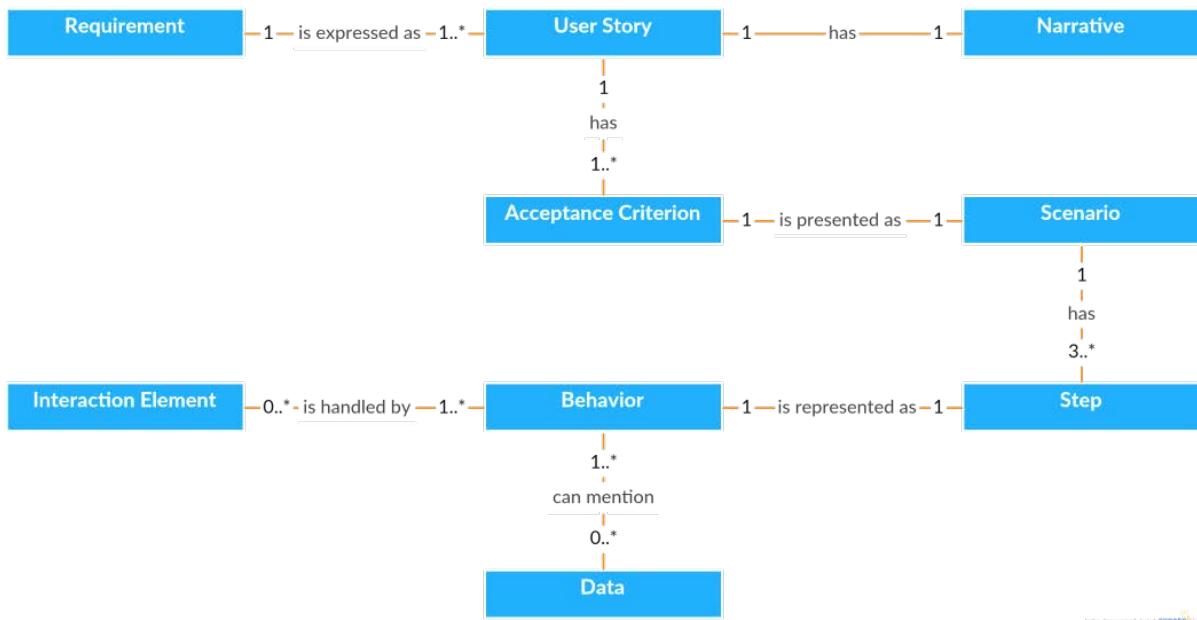


Figure 11. Conceptual Model for testable requirements.

The scenario-based approach supported by the mentioned ontology is focused on functional requirements. As we have stated before, a functional requirement defines statements of services that the system should provide, how the system should react to particular inputs and how the system should behave in particular situations. To assure that the system behaves properly, requirements should be expressed in a testable way. Figure 11 presents the conceptual model that explains how testable requirements are formalized in our proposed approach. A requirement is expressed as a set of User Stories (US) as in the template proposed by North (North, 2017) and Cohn (Cohn, 2004) and presented in chapter 2. User Stories are composed by a Narrative and a set of Acceptance Criteria. Acceptance Criteria are presented as Scenarios and are composed by at least three main Steps (“Given”, “When” and “Then”) that represent Behaviors which the system can answer. Behaviors handle actions on Interaction Elements on the User Interface (UI) and can also mention examples of data that are suitable for testing them. These concepts and rules are defined as classes and axioms in the ontology that will be detailed in the next chapter.

3.1.1. Target Stakeholders

Many stakeholders are typically involved in the development of interactive systems. Table 3 summarizes their typical activities when modeling interactive system and the benefits they can get from using our proposed approach.

Stakeholders	Activity	Benefit
Client / User	Define business and user requirements.	Requirements and automated acceptance testing implemented in a natural and high-level language.
Product Owner and Business Analyst	Write User Stories and define the business model.	A reliable and consistent compatibility between User Stories and business models.
Requirements and Testing Analyst	Write and format User Stories and help to design task models.	A common and standard vocabulary for writing and formatting User Stories.
UI Designer	Design task models and UI prototypes.	A reliable and consistent compatibility between task models and UI prototypes.

Table 3. Target stakeholders of the approach.

3.2. Multiple Views of the Approach

Our approach describes user requirements modeled in a behavior perspective for interactive systems. To illustrate its operationalization, we have defined a micro-process where are represented activities to reach Production and Quality Assurance goals. Figure 12 illustrates User

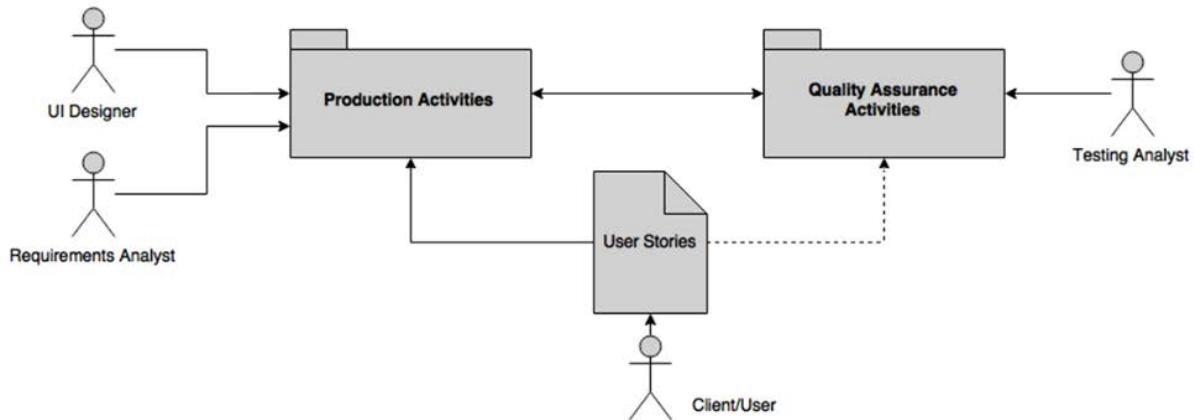


Figure 12. Overall view of the approach.

Stories supporting both Production Activities (which lead to the production of artifacts) and Quality Assurance Activities (aimed at assessing the artifacts produced during the development process). Clients and Users provide the main source of information for the User Stories which will be employed as inputs for Requirements Analysts and User Interface (UI) designers during the Production Activities, as well as for Testing Analysts during the Quality Assurance Activities. Testing Analysts are in charge of building test cases and assessing the artifacts. As testing scenarios are not always explicit in the User Stories told by Clients and Users, Testing Analysts must complement such stories with representative test cases. That is the reason by which we signalize User Stories supporting Quality Assurance Activities with a dotted line. The roles were highlighted separately in the figure in order to emphasize that the activities along the process will require different skills.

The overall view of the approach presented in Figure 12 can be split in two more detailed views: architectural and workflow views. Both of them encompass the same elements, but in different views. Whilst Figure 14 provides a workflow view of activities that have been grouped in Figure 12, Figure 13 highlights the major components and their interactions to accomplish requirements and testing specification.

3.2.1. Architectural View

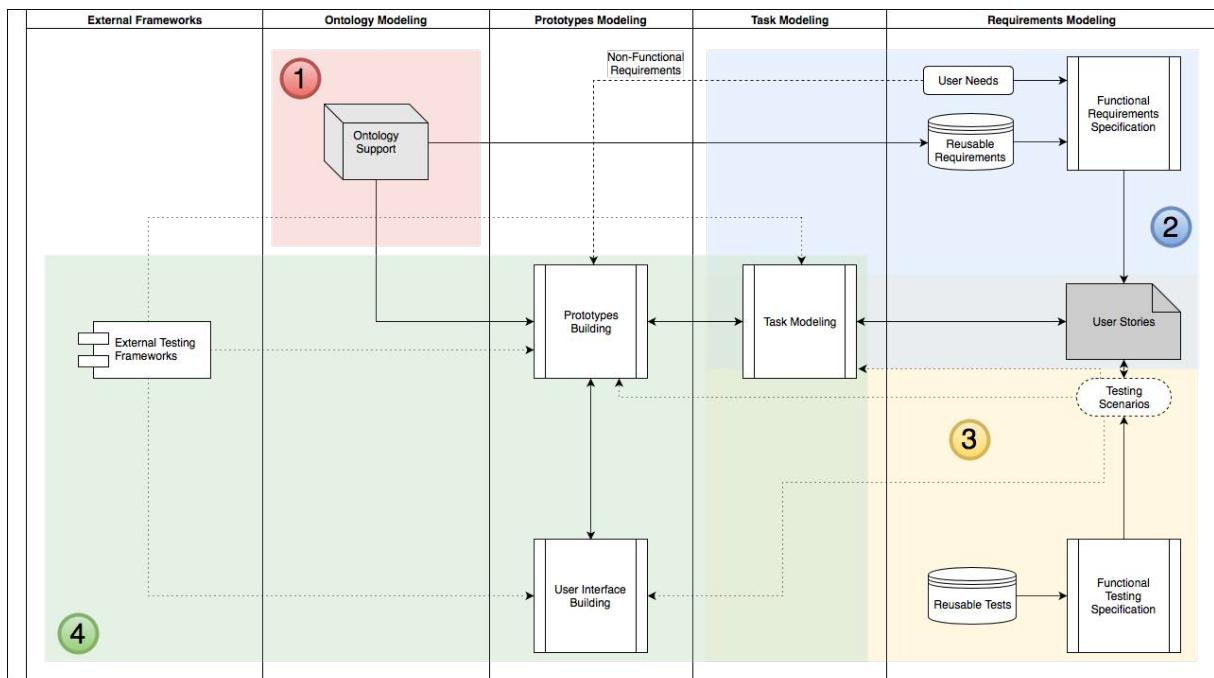


Figure 13. Architectural view of the approach.

In Figure 13, the architectural model is divided in 5 main groups of components: Requirements Modeling, Task Modeling, Prototyping Modeling, Ontology Modeling, and other technical components related to the use of external frameworks to support testing activities. The model is a high-level representation of elementary components of processes, input and output artifacts, and data repositories. Transitions between these elements are represented through direct, optional and/or navigational links.

From the Requirements Modeling perspective, User Needs are grouped with a specific icon to signalize that these needs can emerge from multiple sources such as business models, laws and

regulations, etc. User Needs can describe functional and non-functional requirements. When describing non-functional requirements, User Needs are used as optional inputs (dotted lines) for prototyping activities. When describing functional requirements, User Needs and the set of Reusable Requirements - that are fed by common interactive behaviors mapped in the ontology - are the inputs for the Functional Requirements Specification. This package of activities produces User Stories that are the resultant artifacts of the requirements specification. However, as mentioned in the previous chapter, User Stories are artifacts that encompass at once the set of users' requirements as well as the acceptance criteria for considering such requirements as "done". The acceptance criteria are presented as Testing Scenarios that actually provides the multiple successful and failure paths for assessing the features. These Testing Scenarios are produced during the Functional Testing Specification that can benefit of Reusable Tests as well. Such base of Reusable Tests can be obtained through previous implemented interactive test cases for a given environment.

As a central artifact in the micro-process proposed, User Stories are then produced as a result of requirements and testing activities and serve as a basis for task, prototype and UI modeling. Concerning Task Modeling activities, User Stories are useful when providing the description of functional requirements and the interactive behaviors in the user point of view. Concerning the modeling of prototypes, task models (and by consequence User Stories) support the design of prototypes that are supposed to evolve to fully-fledged User Interfaces. The set of activities that supports such a design is packed in Prototypes Building and User Interface Building packages. As the components of prototyping, task modeling, and the writing of User Stories constitute processes that are dynamic and iterative, they have a bidirectional flow in the architecture representation once they mutually contribute to the development of each other.

The Ontology Support represents the component by which we provide the aforementioned ontology. This ontology describes, among other concepts, behaviors that users perform when interacting with a user interface, besides the correspondent UI elements that support each behavior. As such, the ontology provides support for both Prototypes Building and the reuse of requirements when specifying Functional Requirements. The Ontology Support is also useful to allow checking the consistency of artifacts that will be produced along the development process, as well as to ensure the automated assessment of such artifacts.

Finally, by using the Testing Scenarios, External Testing Frameworks are employed to support automated testing activities in the three target artifacts: task models, prototypes and final UIs. Given development teams can choose the artifacts that will be under testing in each iteration, the optional paths to test them are indicated by dotted lines.

3.2.2. Workflow View

In Figure 14, we have a detailed workflow of the micro-process we have designed for running our approach. In this micro-process, User Stories told by Clients/Users support Production Activities being the main input to produce Scenarios, UI Prototypes and Task Models. User Stories also support Quality Assurance Activities guiding the test of artifacts. For that, Client/Users provide User Stories to the Production Activities which are leaded by Requirements Analysts and UI Designers. These last two roles conduct the process of producing artifacts to be tested by Testing Analysts using Testing Scenarios. Such artifacts can be targets to Clients and Users perform their Acceptance Tests as well.

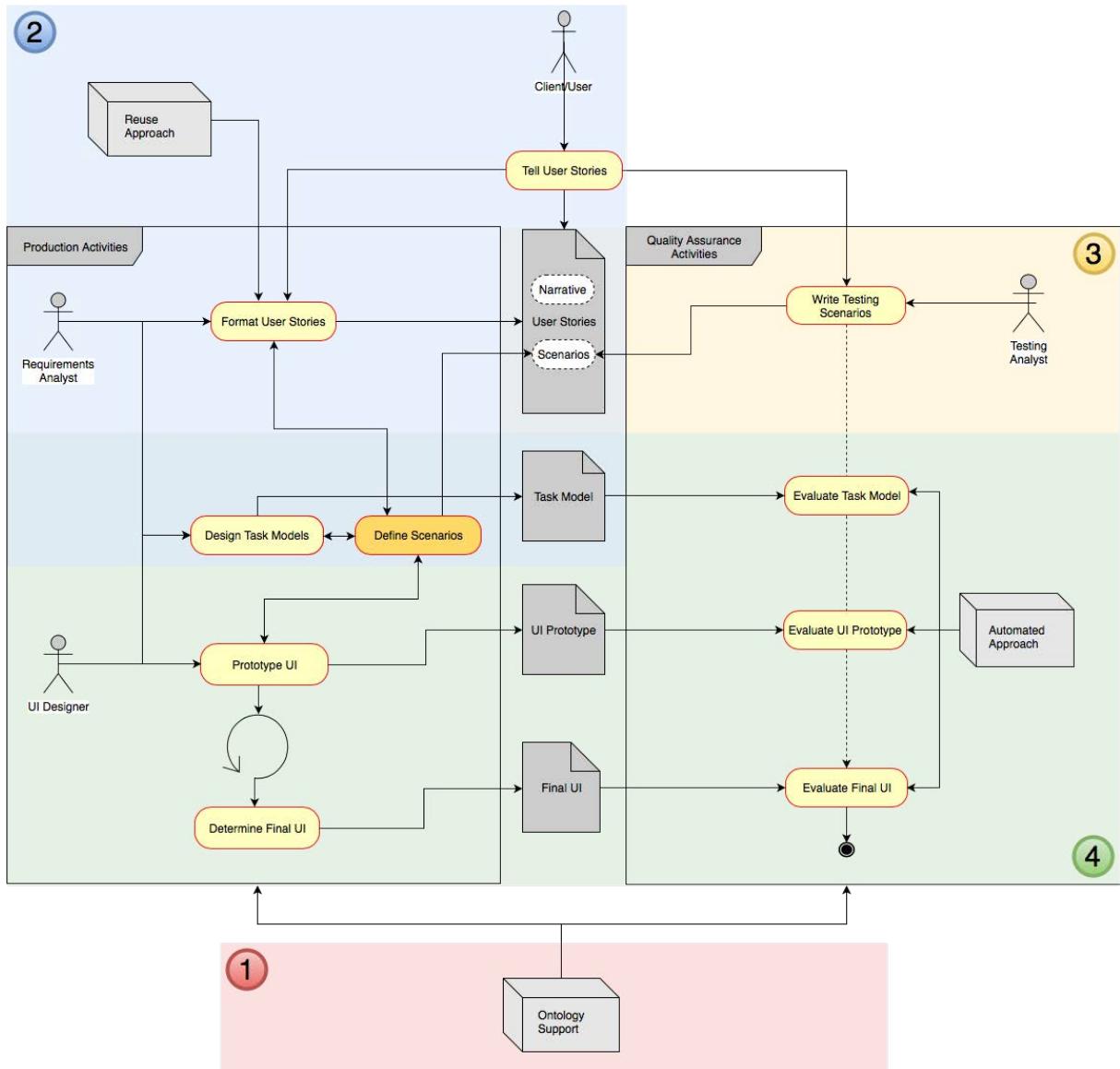


Figure 14. Workflow view of the approach.

Following this, User Stories are then formatted by Requirements Analysts looking for two goals: assuring testability and non-ambiguous descriptions, as well as providing reuse of scenarios in the User Stories. For that, Scenarios are defined in order to identify tasks that users should accomplish using the system. Task Models designed from such Scenarios support a design cycle of successive UI Prototypes and User Interfaces that are produced along system implementation. Prototypes are refined until the Final User Interface can be set. These last activities are conduct by both Requirements Analysts and UI Designers. Notice how Scenarios play a central role in the approach.

Quality Assurance Activities are conducted by Testing Analysts in order to check and verify all the artifacts produced during Production Activities. We are in this case especially interested in testing Scenarios, UI Prototypes and Task Models, as well as the Final User Interface, in an integrated way, in order to ensure consistency between them throughout the development process. Therefore, automated testing frameworks like Webdriver, JBehave and JUnit are used to accomplish these activities, running directly on the artifacts that compose the requirements specification and providing a genuine “live” documentation.

All this mechanism is supported by the use of our ontology that describes concepts used by platforms, models and artifacts that compose the design of interactive systems. The aim is to provide a wide description of elements (and its behaviors) that can be used to build UIs for specific environments. We have initially defined ontologies for the web and mobile platforms and associated the most common behaviors that each element can answer. These behaviors were described using a natural language vocabulary, useful later to specify steps of Scenarios to set actions in these elements.

One of the advantages of the approach is the possibility to reuse scenarios in several systems sharing the same business model. An effective way to provide that is to map steps that compose specific scenarios. Steps are easily reused to build different behaviors in different scenarios. Our approach proposes a set of pre-implemented common steps which perform actions in specific elements on the UI. These actions are described in our ontological model in the next chapter.

Notice lastly that the prototyping of UIs and the modeling of both task models and User Stories are independent activities, i.e. they can be performed individually and, even though they mutually contribute to development of each other, the process does not intend to automate their generation. It means that especially concerning the user interface design, this approach is not supposed to generate prototypes or even final UIs, but rather allow they can be design separately then be tested in order to check their consistency and adherence to the requirements. The designer however can base the design on the ontology description of supported behaviors in order to design prototypes that are promptly consistent with the behavior expected for each interaction element.

3.2.3. Alternatives for Performing the Approach

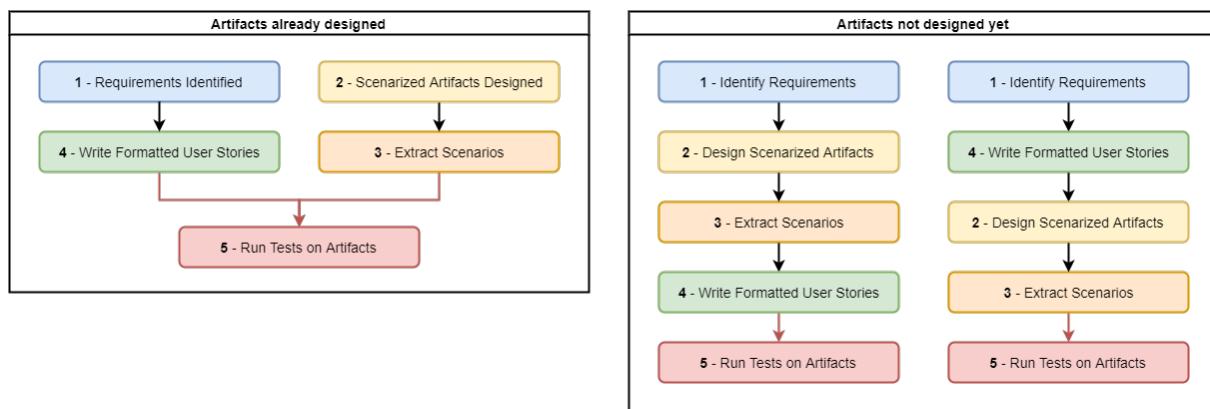


Figure 15. Alternatives for performing the approach.

By looking at the possible project stages in which our approach could be applied to, we have identified two common situations. Figure 15 illustrates these alternatives. The first situation we identified (represented on the left side of Figure 15) concerns the case where our approach will be implemented when the project is running, and artifacts have already been designed (2). If the target artifacts for testing have already been designed, our approach can be used to assess such artifacts, indicating where they are not in accordance with the specified requirements. In this case, requirements are supposed to be already identified (1), so we can directly write our User Stories from these requirements (4), and likewise extract scenarios from the scenarized artifacts (3). When doing that, tests will be ready for running (5).

The second situation (represented on the right side of Figure 15) refers to a project in the beginning, where no artifacts have been designed yet. If the target artifacts have not been designed yet, by using our ontology, they can be modeled in a consistent way since the beginning, taking into account the possible interactions supported by each interaction element on the UI. To this project situation, we could follow sequential steps that include: (1) identify the requirements, (2) design the scenerized artifacts from these requirements, (3) extract scenarios from these artifacts, (4) write our formatted User Stories based on the extracted scenarios, and finally (5) run tests on the artifacts. Alternatively, we can perform the activity 4 (write formatted User Stories) before the activity 2 (design scenerized artifacts). It means that depending on the characteristics of the project, either the User Stories can support the design of the artifacts, or the artifacts (by means of their extracted scenarios) can support the writing of User Stories.

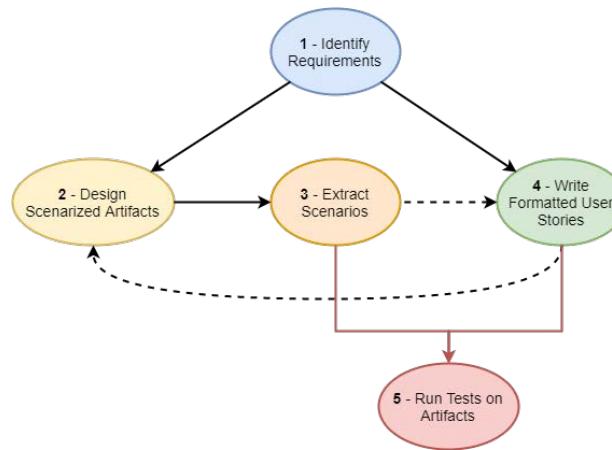


Figure 16. The graph of options for performing our approach.

Figure 16 illustrates the resultant graph of options considered. Notice that solid lines indicate mandatory activities, i.e. we must either design scenerized artifacts (2) or write formatted User Stories (4) only after having identified requirements (1); extract scenarios (3) only after having designed scenerized artifacts (2); and run tests on scenerized artifacts (5) only after having extracted scenarios (3) and written formatted User Stories (4). The optional paths represented by the dotted lines indicate the alternatives shown on the right side of Figure 15, i.e. we can either use the extracted scenarios (3) to support the writing of formatted User Stories (4) or use the formatted User Stories (4) to support the design of scenerized artifacts (2).

The high-level operationalization of the approach is made up in four main groups of activities that are pinpointed in Figure 13 and Figure 14 by numbers as follows:

- (1) definition of the ontology,
- (2) writing testable User Stories,
- (3) adding test scenarios, and
- (4) multi-artifact testing.

In the next section (3.3), we detail how we start writing User Stories (group 2) and how we add test scenarios to those stories (group 3) by means of an illustrative case study. These two groups of activities in our approach are supported by the definition of the ontology (group 1) that will be explored in chapter 4. Finally, our actual strategy to conduct automated testing on multiple artifacts (group 4) will be presented in the section 3.4 and explored along the thesis.

3.3. A Case Study in a Nutshell

To illustrate the operationalization of our approach, we have proposed a generic case study in the flight tickets e-commerce domain. This study acts a proof of concept for our approach and will be used along the next chapters to show how the approach can support the assessment of our target artifacts, i.e. task models, prototypes and final UIs. This case study was chosen because it is easily comprehensible, and it represents a common activity for most of the readers nowadays. For the study, we have considered only two actors involved, a user and an airline company. We have based on a generic flow of activities including users searching their flights, picking them up from a list of results, and then confirming his/her choices by providing passengers and payment data. In theory, this generic flow of activities could be applied to any airline company selling tickets on the web.

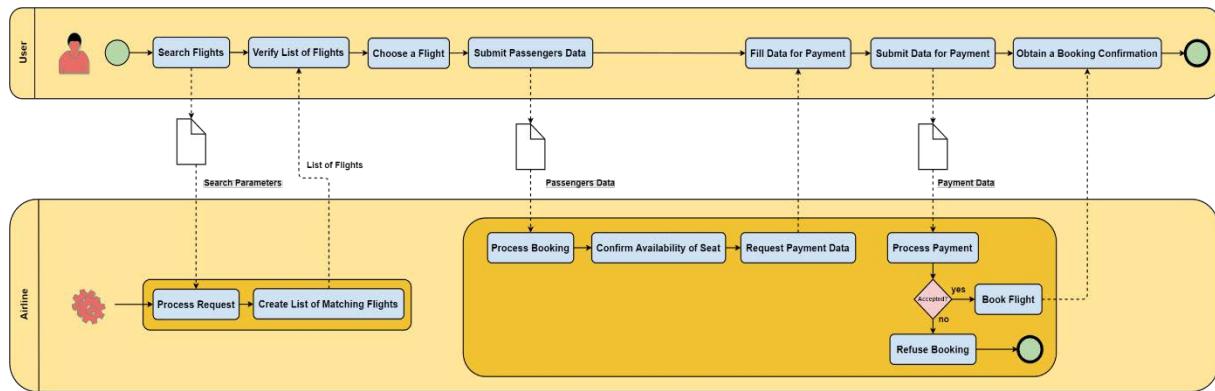


Figure 17. Business Process Model for the flight ticket e-ticket domain.

Figure 17 presents the business model for this case study, using the Business Process Model and Notation (BPMN) (*Business Process Model And Notation™ (BPMN™)*, 2011). At the top, in the first lane, we have the set of activities performed by users. In the second lane, we have the set of activities performed by the airline company. In a first moment, the set of activities performed by the airline company could be made either manually or in an automated way (using a software system). For this study, we are assuming that the choice is to conduct these activities in an automated way, using a web software system. The set of functional requirements assumed by the system is described below through a narrative scenario:

The user starts the process by conducting a search of flight based on his desired parameters like origin and destination, dates, number of passengers, etc. This set of parameters is then submitted to the airline system that will process the re-quest and creates a list of matching flights. The list of flights is then returned to the user that verify this list and chooses a flight that better suit his needs. After choosing the desired flight, the user provides all passengers data to the airline system that will process the booking. Thereby, the system confirms the availability of seats and request user to provide payment data. After the user filled in the forms with bank account details and confirmed the payment, the system will process the transaction. If the payment is accepted, then the booking is completed, the user obtains a booking confirmation and the process finishes. If the payment is declined, then the booking is refused, and the process finishes as well.

The online booking process described above is basically divided into 3 main sub processes: searches of flights based on a provided set of data, the selection of the desired flight(s) in a list of flights resultant from the search, and finally providing passenger and payment data to conclude

the booking. We have selected the two first processes for this case study as they are the most interactive ones and represent the main source of cognitive efforts from users and designers. The third sub process is basically a data providing form, so it is not so relevant to demonstrate the concepts we want, even though the whole process can be supported by this approach.

In the following subsections, we use the present case study to illustrate the groups of activities 2 (writing testable User Stories) and 3 (add test scenarios) pinpointed in Figure 13 and Figure 14.

3.3.1. Writing Testable User Stories

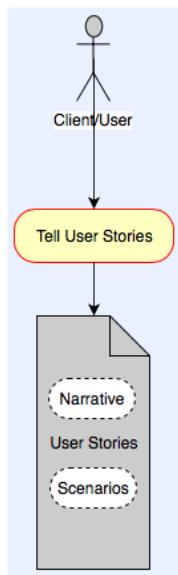
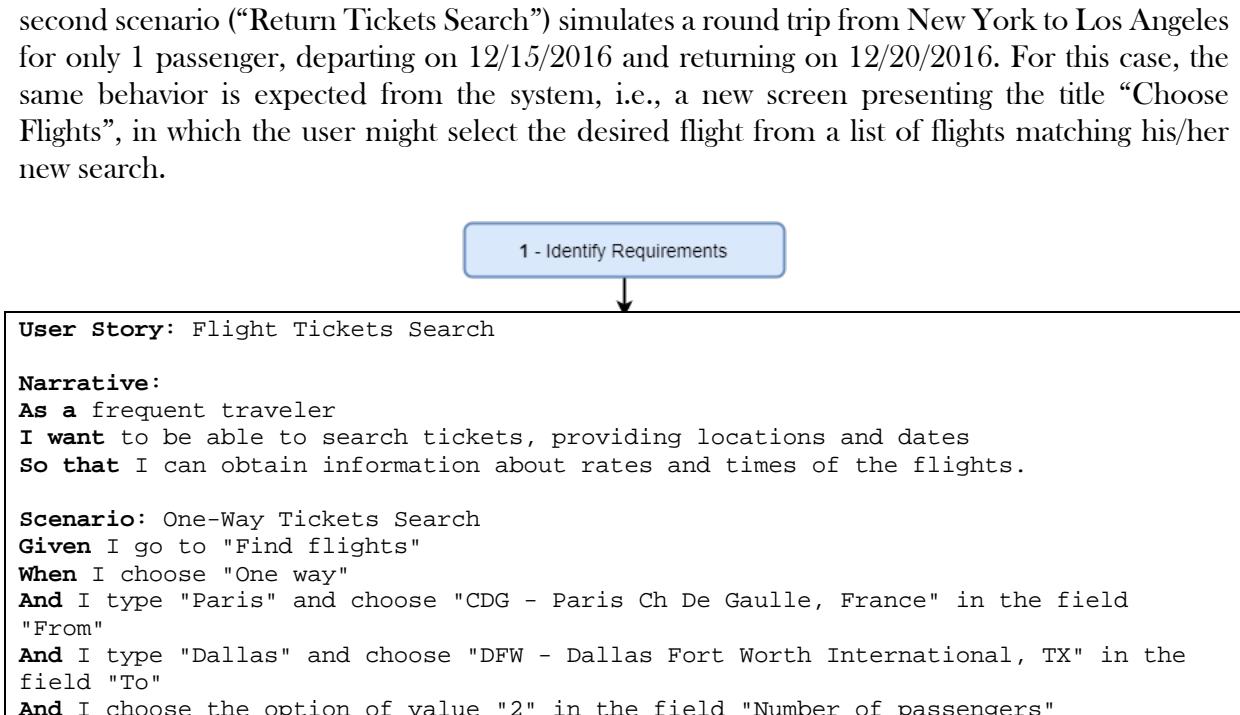


Figure 18. Activity of telling User Stories

Hereafter, we present two User Stories with their respective scenarios to describe and test the features of our case study. Such User Stories simulate the identification of requirements in stories told by users according to the activity in our workflow (Figure 18). This is a first attempt of getting testable User Stories (1 - Identify Requirements) once they will be still formatted afterwards to fit the interactive behaviors described in the ontology. These stories focus on the process of searching flights of our illustrative case study, with a narrative describing the role involved with the history ("As a"), the feature that this history describes in the user's point of view ("I want"), and finally the benefit (business value) that this feature brings to the user in terms of business goals ("So that").

The first story presents the procedure for searching flights in which the user should provide at least: the type of ticket he wants (one-way or round trip), the airport he wants to depart from and arrive at, the number of passengers in the trip, and finally the date of departure and return. In the first scenario ("One-Way Tickets Search"), a typical search of tickets is presented concerning a one-way trip from Paris to Dallas for 2 passengers on 12/15/2016. According to the business rule, the expected result for this search is a new screen presenting the title "Choose Flights", in which the user might select the desired flight from a list of flights matching his/her search.



```

And I set "12/15/2016" in the field "Depart"
And I click on "Search"
Then will be displayed "Choose Flights"

Scenario: Return Tickets Search
Given I go to "Find flights"
When I choose "Round trip"
And I type "New York" and choose "NYC - New York, NY" in the field "From"
And I type "Los Angeles" and choose "LAX - Los Angeles International, CA" in the field "To"
And I choose the option of value "1" in the field "Number of passengers"
And I set "12/15/2016" in the field "Depart"
And I set "12/20/2016" in the field "Return"
And I click on "Search"
Then will be displayed "Choose Flights"

```

The second history focuses on the process of choosing a flight in a list of available flights. The scenario “Select a diurnal flight”, using the scenario “One-Way Tickets Search”, simulates the selection in the list of available flights, a couple of diurnal flights, the AA6557 and the AA51. For this case, the behavior expected from the system is the presentation of a new screen with the “Optional log in” message, indicating the user is able to login in order to proceed to the booking, filling the passengers and payment data, which is in line with both business and task models.

```

User Story: Select the desired flight

Narrative:
As a frequent traveler
I want to get the list of flights and their rates and times
So that I can select the desired flight after a search of available flights.

Scenario: Select a diurnal flight
One-Way Tickets Search
Given "Flights Page" is displayed
When I click on "Flights" referring to "AA flight 6557, AA flight 51"
Then "Optional log in" is displayed

```

3.3.2. Adding Testing Scenarios

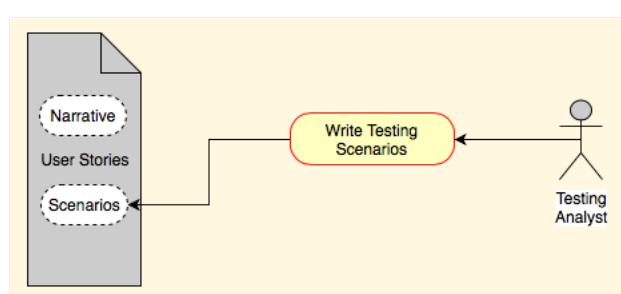


Figure 19. Activity of creating testing scenarios

descriptions of functionalities as well as the potential set of tests to verify the correct implementation of the requirements. Functional testing is the leading element of the acceptance level and is used to check expected outcomes when pre-defined inputs are provided to the system.

Below we present two testing scenarios: “Search for flights more than one year in advance” and “Search for a return flight before a departure flight”, that will be added to the User Story “Flight Ticket Search”. They present specific business rules (and their tests) in the flight-booking domain. The expected outcome in both cases is the impossibility of searching flights.

Test cases are represented as Testing Scenarios in our approach (Figure 19). They specify potential error situations related to the scenarios already defined to set requirements. Testing scenarios are the component responsible for describing the situations in which the system should be verified, covering, as deeply as possible, the largest set of features. Thereby, requirements scenarios and testing scenarios compose the User Stories, providing in the same artifact,

It is important to notice that testing scenarios describe a test procedure that may be generic regarding the data demanded to run a test case. When test data are added to a test procedure then it becomes a test case. This fact gives us the opportunity to write a single test procedure once and reuse it, in order to generate multiple test case, based on multiple test data. The two examples below are already specified with test data, so they can also be seen as test cases.

```
Scenario: Search for flights with more than one year in advance
Given I go to "Find flights"
When I choose "One way"
And I type "Paris" and choose "CDG-Paris Ch De Gaulle, France" in the field "From"
And I type "Dallas" and choose "DFW-Dallas Fort Worth International, TX" in the
field "To"
And I choose the option of value "1" in the field "Number of passengers"
And I try to choose "12/15/2017" referring to "Depart"
Then the system should not allow performing this task

Scenario: Search for a return flight before a departure flight
Given I go to "Find flights"
When I choose "Round trip"
And I type "New York" and choose "NYC-New York, NY" in the field "From"
And I type "Los Angeles" and choose "LAX-Los Angeles International, CA" in the
field "To"
And I choose the option of value "1" in the field "Number of passengers"
And I try to choose "12/15/2016" referring to "Depart"
And I try to choose "12/10/2016" referring to "Return"
Then the system should not allow performing this task
```

3.4. Strategy for Testing

Our strategy for running tests on multiple artifacts is shown in Figure 20. The figure illustrates User Story scenarios being used to ensure consistency in our target artifacts (task models, UI prototypes and final UIs). Therein are exemplified five steps of scenarios being tested against equivalent tasks in task model scenarios, and interactive elements in UI prototypes and final UIs. In the first example, the step “When I select ‘<field>’” has found an equivalent correspondence with the task “Select <field>” in the task model scenario. Such an equivalence is due to the fact that the step and the task represent the same behavior, i.e. selecting something, and both of them are placed in the first position in their respective scenario artifacts. The interaction element “field” that will be affected by such a behavior will be assessed on the UI prototype and on the final UI. In both artifacts, such a field has been designed with a CheckBox as interaction element. The semantics of the interaction in CheckBoxes is compatible with selections, i.e. we are able to select CheckBoxes, so the consistency is assured.

The same is true in the example with the second step (“When I click on ‘<field>’”). There is an equivalent task “Click on <field>” in the same second position in the task model scenario, and the interaction element “Button”, that has been chosen to address this behavior in both the UI prototype and the final UI, is semantically compatible with the action of clicking, thus the consistency is assured as well. In the third example, the step “When I choose ‘value’ referring to ‘field’” is also compatible with the task “Choose <field>” in the task model, and with the interaction elements “DataChooser” and “Calendar”, respectively in the UI prototype and in the final UI. Notice that, despite being two different interaction elements, “DataChooser” and “Calendar” are equivalent in their semantics of behaviors supported, i.e. both of them support the behavior of choosing values referring to a field.

The example provided with the fourth step (“When I click on ‘<field>’”) illustrates an inconsistency being identified. Therein, despite existing an equivalent task in the task model scenario, the interactive elements that have been chosen to address this behavior (“TextInput” in

the UI prototype and “TextField” in the final UI) are not compatible with the action of clicking, i.e. such kind of interaction element does not semantically support such an action. The semantics of TextInputs (or TextFields) is receiving values, not being clicked. Such an example is provided with the fifth step (“When I set ‘value’ in the field ‘<field>’”). For this step, the consistency is assured because TextInputs and TextFields support the behavior of having values being set on them. All this semantic analysis is supported by the use of the ontology.

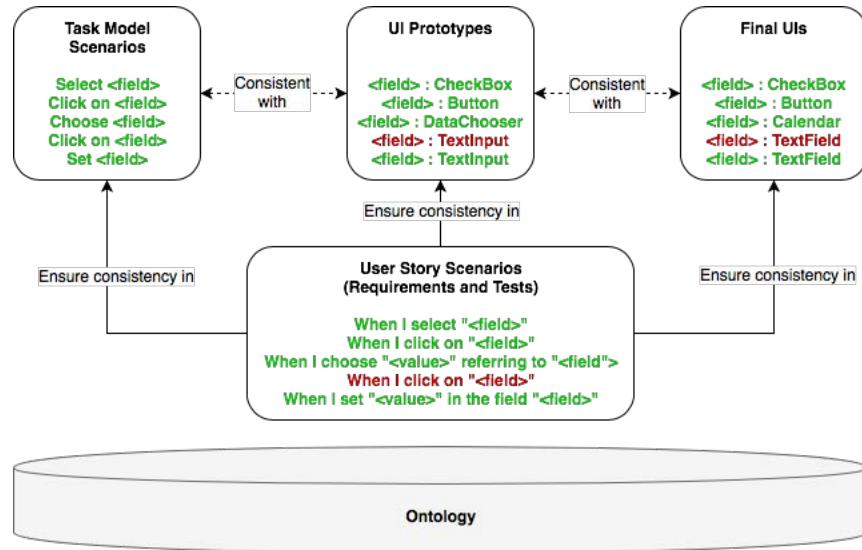


Figure 20. Our strategy for testing.

The present strategy we defined for testing allows us tracking some key elements in the artifacts and check whether they are consistent with the user requirements. By simulating user's actions, our approach also allows that interactive prototypes and final UIs are directly tested by the users' acceptance criteria in order to ensure that the artifacts are consistent with the user requirements. Resuming the classification in groups of artifacts we set up in the beginning of this chapter (section 3.1), when assessing early artifacts from the first group, we are actually complying with the verification aspect of software testing, once by definition, we are comparing the requirements baseline with the successive refinements descending from it (i.e. the artifacts) in order to keep these refinements consistent with the requirements baseline. When assessing late interactive artifacts from the second group (such as final UIs), we are also complying with the validation aspect of software testing, once these artifacts are tested simulating the user's actions, thus checking if the software product satisfies or fits the intended use according to the user's acceptance criteria.

In the current literature, especially when verifying software artifacts, the term “test” is usually not employed under the argument that such artifacts cannot be “run”, i.e. executed for testing purposes, so in practice they are just manually reviewed or inspected. As within our approach we succeed automatically running our target artifacts for assessing their consistency with user requirements, we actually provide the “test” component for the verification of artifacts in the software development. We consider this is a big step towards the automated testing (and not only the manual verification) of software artifacts by means of a consistent approach allowing fully verification, validation, and testing (VV&T) (Engel, 2010). The complete testing strategy will be explored in chapters 5, 6 and 8 to show how we perform tests for checking the consistency, thus verifying and validating, the set of our target artifacts.

3.5. Conclusion

The present chapter presented the motivation and the inner background for proposing a scenario-based approach for testing multiple artifacts. This chapter is aimed at providing a view at glance of the approach. The instantiation of the approach should be tuned according to the very specific artifacts target for testing and it is detailed latter on in chapter 5 (for task models) and in chapter 6 (for user interface prototypes). Nonetheless, it is interesting to notice that the implementation of this scenario-based approach relies on some basic premises as follows:

- (i) *To adhere to a model-based approach for describing artifacts produced along the development process.* This is due to the fact of our approach is intended to assess artifacts resultant from modeling activities.
- (ii) *Teams must be willing to adopt the template for User Stories as well as the vocabulary proposed in the ontology.* This is due to the need of formalization of user requirements for testing. As a certain level of adherence to a template is required, this could eventually be an issue for development teams which already use other approaches for requirements specification.
- (iii) *Artifacts and the user interface under testing must comply with the UI-supported set of interactive behaviors described in the ontology.* This is due to the fact that the ontology encompasses an extensive, but fixed number of interaction elements and behaviors supported by web and mobile user interfaces.
- (iv) *Tests must be carried out by our set of tools.* This is due to the fact that our strategy for testing is only implemented in our set of tools, so they must be used to perform the tests on the target artifacts and on the final UI.

By tackling these challenges, the use of the proposed approach could promote a set of advantages as follows:

- requirements and tests in a natural and high-level language,
- independence for testing artifacts,
- independence of software development processes,
- no need to prepare artifacts for testing,
- interactive behaviors kept the same regardless the application domain,
- plurality of interaction elements modeled by the ontology,
- fine-grained testing coverage, and
- the use of data-independent scenarios.

From the stakeholders' point of view, this approach can address multiple concerns related to requirements specification. For clients and users, requirements and the acceptance testing have the benefit of being specified and implemented in a natural and high-level language. The benefits of non-technical stakeholders' involvement in requirements specification are largely known in the literature (Bano and Zowghi, 2013). They include reducing requirements misunderstandings, besides providing faster feedback and more accurate acceptance conditions.

For Product Owners and Business Analysts, which write User Stories and define the business model, the benefit would be a reliable and consistent approach for checking the compatibility between User Stories and business models. For Requirements and Test Analysts, a common and standard vocabulary for writing and formatting User Stories would help to improve communication between the business people and the development team. Being a single artifact encompassing both requirement specification and acceptance testing, User Stories also tackle the typical problem of alignment between requirements and tests (Hotomski, Charrada and Glinz,

2017). Finally, UI designers would benefit from a reliable and consistent approach for checking the compatibility between task models and UI prototypes in different levels of refinement.

The next chapter will describe and present our supporting ontology, followed by two chapters describing in detail the strategy presented here for modeling and testing our target artifacts: task models (in chapter 5) and UI prototypes (in chapter 6).

3.6. Resultant Publications

Silva, T. R. (2016). Definition of a Behavior-Driven Model for Requirements Specification and Testing of Interactive Systems. In: 2016 IEEE 24th International Requirements Engineering Conference (RE 2016), pp. 444-449. IEEE. DOI: <http://doi.org/10.1109/RE.2016.12>. (Silva, 2016)

Silva, T. R., Hak, J. L. & Winckler, M. (2016). Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development. In: 6th International Working Conference on Human-Centred Software Engineering, and 8th International Working Conference on Human Error, Safety, and System Development (HCSE 2016 and HESSD 2016), pp. 86-107, vol. 9856. Lecture Notes in Computer Science, Springer International Publishing. DOI: http://doi.org/10.1007/978-3-319-44902-9_7. (Silva, Hak and Winckler, 2016b)

Silva, T. R., Hak, J. L. & Winckler, M. (2016). An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development. Complex Systems Informatics and Modeling Quarterly, 1 (7), pp. 81-107. DOI: <http://doi.org/10.7250/csimg.2016-7.05>. (Silva, Hak and Winckler, 2016a)

Silva, T. R. & Winckler, M. (2016). Towards Automated Requirements Checking Throughout Development Processes of Interactive Systems. In: 2nd Workshop on Continuous Requirements Engineering, 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ). CEUR-WS.org. (Silva and Winckler, 2016)

Chapter 4

Towards an Ontology for Supporting GUI Automated Testing

Summary

This chapter presents the ontological approach we have developed for specifying interaction and supporting UI automated testing. The aim of the ontology described in this chapter is to support the assessment of interactive systems, providing a common and consistent description of elements that compose the semantic of interaction between users and systems in a web and/or mobile environment.

The ontology aims to support testing automation of interactive systems specified using a scenario-based approach, covering UI concepts in both presentation and dialog aspects. For the presentation layer, we have modeled the semantics of several web and mobile UI elements. For the dialog layer, we have modeled the semantics of User Stories as a State Machine. Such models have allowed us to provide a semantically consistent catalog of interactive behaviors that can be used for automating the test of UIs in different levels of abstraction.

The first subsection of this chapter discusses the related approaches that inspired this ontology, including a comparative analysis of how each approach has contributed to the development of the ontology and the contribution it provides in different aspects of modeling. The second subsection presents the detailed description of the ontology, covering its technical OWL specification for classes, individuals, datatypes, as well as object and data properties. Results of our ontology validation are also presented by demonstration of its correctness through an automated consistency checking. Finally, the third and last subsection presents limitations and perspectives concerning the use of the ontology for testing purposes.

In chapter 3, we have presented the big picture of the approach being proposed in this thesis where we pointed out the use of an ontological support for both Production and Quality Assurance activities. The ontology we proposed for such support is motivated by our previous experience as requirements/test engineers in industry, developing e-Government web applications in the biggest public software development company in Brazil. During more than five years implementing GUI testing, we have observed certain patterns of low-level behaviors that are recurrent when writing BDD Scenarios for testing functional requirements with the User Interface (UI). Besides that, we could also observe that User Stories specified in natural language often contain semantic inconsistencies. For example, it is not rare to find Scenarios that specify an action such as a selection to be made in semantically inconsistent widget such as a Text Field. These observations motivated us to investigate the use of a formal ontology for describing pre-defined behaviors that could be used to specify Scenarios that address interactions with UIs. On one hand, the ontology should act as a taxonomy for terms removing ambiguities in the description. On the other hand, the ontology would operate as a common language that could be used to write tests that can be run on many artifacts used along the development process of interactive systems. However, it is important to notice that the ontology does not propose a new

language for describing UIs, but rather a direct mapping between languages for describing the interface and common behaviors for testing.

4.1. Related Approaches

Computational ontologies (Guarino, Oberle and Staab, 2009) come to play as a means to formalize the vocabulary and the concepts used in User Stories, Scenarios and UIs. Without a common agreement on the concepts and terms used it would be difficult to support the assessment of user requirements. Especially in the context of User Interface design, some approaches have tried to define languages or at least a common vocabulary for specifying UIs for interactive systems. Useful attempts include abstractions for describing interactive systems with components that compose the presentation of a User Interface (Calvary *et al.*, 2002, 2003; Puerta and Eisenstein, 2002; Fierstone, Dery-Pinna and Riveill, 2003; Limbourg *et al.*, 2004; Farooq Ali, Pérez-Quiñones and Abrams, 2005; Pullmann, 2017), or even the dialog for implementing the system behavior (Calvary *et al.*, 2002, 2003; Winckler and Palanque, 2003; Winckler *et al.*, 2008; Barnett, 2017). However, the problem raised in such approaches is that they do not provide a formal model for both presentation and dialog aspects, thus not allowing the specification of behaviors for UI testing, i.e. there is not a common pattern for such a specification. Such approaches work much more as a meta-model, letting the formalization of their concepts to be specified or implemented by third frameworks.

4.1.1 Compared Overview

The contribution of the ontology proposed in this chapter can be analyzed comparing it with other methods and languages from which it borrows concepts. This analysis is presented in Table 4 for Cameleon Framework (Calvary *et al.*, 2002) and UsiXML (Limbourg *et al.*, 2004), as well as for W3C MBUI Glossary (Pullmann, 2017) and SWC (Winckler and Palanque, 2003). The Cameleon Reference Framework decomposes user interface design into a number of different components that seek to reduce the effort in targeting multiple contexts of use (Calvary *et al.*, 2002). These components are Task-Oriented Specification, Abstract UI, Concrete UI and Final UI. The ontology has been built based on this decomposition, with high-level description of tasks being modeled as a task-oriented specification (based on notation such as CTT and HAMSTERS). UsiXML implements the Cameleon Framework in an XML specification, which allows us operating these concepts in the ontology. SWC adds the dialog component for the Cameleon/UsiXML specification allowing us specifying transitions and adding navigation to the User Interface. Finally, W3C MBUI Glossary contributes establishing the common vocabulary used by the other methods and languages. This common vocabulary is used to describe elements in the ontology.

	Concept	Mapping in the ontology
Cameleon and UsiXML	Task-Oriented Specification: This concept describes the tasks that the user and the system carry out to achieve the application's objectives. The tasks are described at a high level that is independent of how these are realized on a particular platform.	Description of Scenario-based concepts , including the modeling of Users Stories and Tasks .
	Abstract UI: This level describes models of the user interface that are independent of the choice of platform and of the modes of interaction (visual, tactile, etc.).	Description of Interaction Elements in the Presentation perspective.

	Concrete UI: This level models the user interface for a given platform, e.g. desktop PC, tablet, smart phone, connected TV and so forth.	Platform concepts are described in the ontology, as well as the list of interaction elements that are supported by each platform (web and mobile).
	Final UI: This level implements the user interface for a specific class of device, e.g. an iPhone, or an Android tablet.	The ontology provides means of reading the set of interaction elements supported by each user interface platform. It allows designing automated testing implementations for specific platforms based on such elements to create concrete graphical widgets.
SWC	Task Model (TM): Tasks and dependencies between tasks.	Description of State Machine concepts . The dynamic behavior of tasks being performed by users and systems are described as Scenario-based concepts .
	Abstract User Interface (AUT): Relationship between logical presentation units (e.g. transition between windows), logical events, abstract actions.	Description of Interaction Elements in the Dialog perspective.
	Concrete User Interface (CUI): States, (concrete) events, parameters, actions, controls, changes on UI dialog according to events, generic method calls, etc.	Description of the Transition triggers in the State Machine that each behavior may perform on the user interface.
	Final User Interface (FUI): “Physical” signature of events, platform specific method calls, etc.	The ontology provides means of reading the set of behaviors supported by each interaction element. It allows designing automated testing implementations for specific platforms based on such behaviors to create concrete class methods for automating the “physical” interaction on the user interface.
W3C MBUI Glossary	It is a glossary of terms recurrent in the Model-based User Interface domain (MBUI). It contains informal, commonly agreed definitions of relevant terms and explanatory resources.	Description and definition of Platform and UI concepts .

Table 4. A compared overview between the ontology and other methods and languages.

4.2. A Behavior-Based Ontology for Interactive Systems

Our ontology for describing interactive systems is based on concepts borrowed from different languages found in the literature. From Camaleon and UsiXML we borrow the concepts of abstract and concrete UIs. Presentation and definition of graphical components come from W3C MBUI. From W3C Web Ontology Language we get concepts for graphical components (behavior and presentation aspects) commonly used to build web and mobile applications, and also the textual representations used to describe how users interact with those graphical components. SWC inspires concepts used to describe the dialog. Like many other approaches (Calvary *et al.*, 2002, 2003; Winckler and Palanque, 2003; Winckler *et al.*, 2008; Barnett, 2017), our description of dialog in the ontology is based on the specification of a classical state machine. Such a reuse of concepts reduced considerably the modeling effort and allowed us to propose an

ontology consistent with well-known approaches for describing both the presentation and the dialog of user interfaces.

The ontology has been modeled in Protégé 5.0. Figure 21 presents the classes of the ontology and their properties divided in 4 wide groups: Platform Concepts, UI Concepts, State Machine Concepts and Scenario-based Concepts. These groups are represented as clouds in the figure. Classes are represented as rectangular boxes, and the relationships between classes (i.e. their Object Properties) are represented by solid lines which include the name of the Object Property and the constraint associated to the relationship. Finally, dotted lines represent a generalization/specialization relationship, i.e. an “is_a” Object Property. For convenience, lines representing relationships that share the same Object Property name and the same constraint were merged to improve the legibility of the image.

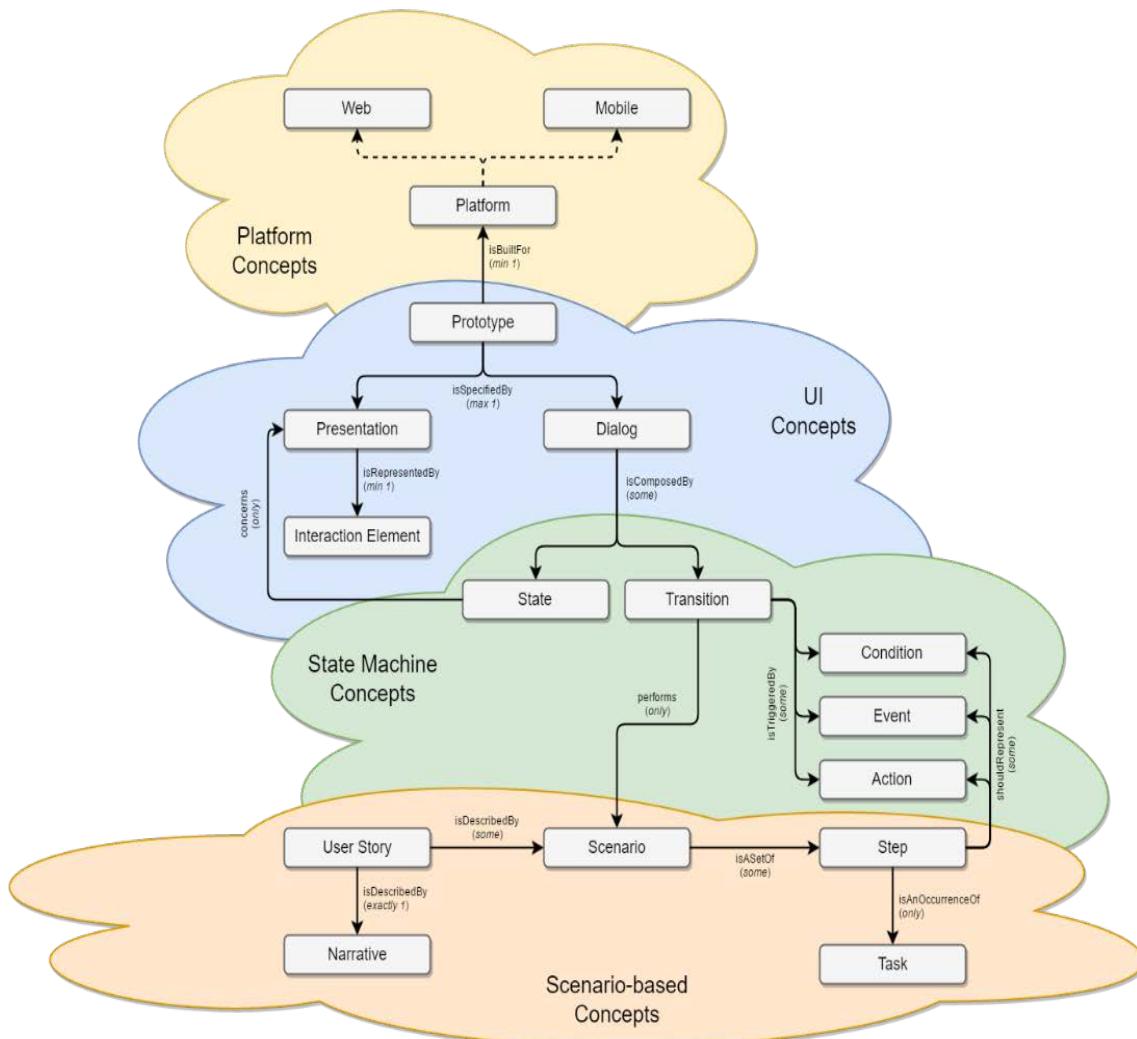


Figure 21. Main classes and their properties in the ontology.

The first group of concepts defines the web and mobile platforms covered by the ontology. The second one encompasses concepts allowing modeling the UI. The classes Dialog, Presentation and Platform model the concept of a Prototype. A Prototype is built for at least one Platform and specified by no more than one Dialog and one Presentation. The third group specifies the State Machine concepts. Therein, a Dialog is composed by States and Transitions, whilst a Presentation, which is represented by at least one Interaction Element, is concerned by

only one State at once. In the fourth group of concepts, the classes Narrative, Scenario, Step and Task model the concept of a User Story. A User Story is described by exactly one Narrative and some Scenarios. A Scenario is a set of Steps and a Step is an occurrence of only one Task. A Step shall represent some Event, Condition or Action which together trigger a Transition in the State Machine. Finally, a Transition performs a given Scenario from the User Stories.

Concepts have been modeled as Classes. Relationships between concepts have been modeled as Object Properties (subtype “relations”). Classes that handle data have such descriptions modeled as Data Properties. As core elements in the ontology, UI Elements and the interactive behaviors have been modeled respectively as Classes and Object Properties (subtype “behaviors”).

In the following subsections, we detail the basic concepts of Object (subsection 4.2.1) and Data Properties (subsection 4.2.3), as well as the four main group of concepts described above: Platform (subsection 4.2.4), UI (subsection 4.2.5), State Machine (subsection 4.2.6), and finally Scenario-based concepts (subsection 4.2.7). The current version of the ontology bears an amount of 677 axioms (being 482 logical axioms), 58 classes, 79 object properties, 16 data properties and 3 individuals. A visual representation of all the concepts can be found at <https://goo.gl/IZqSJ0> and its complete specification in OWL can be found at <https://goo.gl/1pUMqp>.

4.2.1 Object Properties

Relationships of individuals in classes are represented as Object Properties (OP). We have classified these properties in “Relations” and “Behaviors”. “Relations” groups conceptual relationships between objects from internal classes, i.e. objects that do not directly address interactive behaviors. “Behaviors”, on the other hand, groups conceptual relationships between interactive behaviors and UI Elements on the UI. Besides these two groups of OPs, we have also modeled two single Object Properties (*allowsUnique* and *allowsMultiple*) to express the relationship between some UI elements and their Data Properties (DP). The “Relations” group is detailed hereafter, whilst the “Behaviors” group will be detailed in the subsection 4.2.6, and the single OPs will be presented in the subsection 4.2.5.

4.2.2 Relations

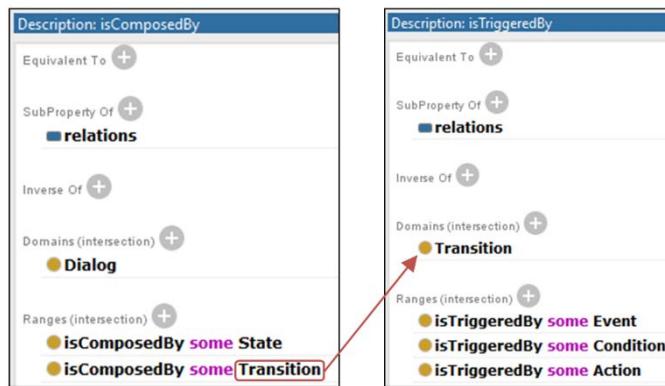


Figure 22. Object Properties *isComposedBy* (left) and *isTriggeredBy* (right).

The sub property “relations” defines the semantic correspondence between internal classes. Table 5 presents the whole set of relationships between objects of internal classes defined in the ontology. The class that drives the property is called Domain Class and the class affected by the property is called Range Class. The Restriction Type adds constraints to the modeled property.

Figure 22 illustrates the relations between elements in the State Machine. As a sub property of Relations, objects from the Dialog class are composed by some States and Transitions. This relationship is described by the property *isComposedBy* (left side of Figure 22). Accordingly, objects from the Transition class are triggered by a sequence of some Conditions, Events and Actions. This relationship is described by the property *isTriggeredBy* (right side of Figure 22).

Domain Class	Object Property	Restriction Type	Range Class
State	<i>concerns</i>	only	Presentation
Step	<i>isAnOccurrenceOf</i>	only	Task
Scenario	<i>isASetOf</i>	only	Step
Prototype	<i>isBuiltFor</i>	min 1	Platform
Dialog	<i>isComposedBy</i>	some	State
	<i>isComposedBy</i>	some	Transition
User Story	<i>isDescribedBy</i>	exactly 1	Narrative
	<i>isDescribedBy</i>	some	Scenario
Presentation	<i>isRepresentedBy</i>	min 1	Interaction Element
Prototype	<i>isSpecifiedBy</i>	max 1	Dialog
	<i>isSpecifiedBy</i>	max 1	Presentation
Transition	<i>isTriggeredBy</i>	some	Event
	<i>isTriggeredBy</i>	some	Condition
	<i>isTriggeredBy</i>	some	Action
Transition	<i>performs</i>	only	Scenario
Step	<i>shoudRepresent</i>	some	Event
	<i>shoudRepresent</i>	some	Condition
	<i>shoudRepresent</i>	some	Action
Mobile	<i>usesAsAMobileElement</i>	some	<UI Element>
Web	<i>usesAsAWebElement</i>	some	<UI Element>

Table 5. “Relations” as Object Properties in the ontology.

4.2.3 Data Properties

Data Properties are used to describe semantically data domains used by each class that handles data. Our ontology has been designed following Ontology Design Principles (Dumontier, 2018), so Datatypes were specified under the standard XSD specification and constraints were defined to restrict the set of data domains applied to each Domain Class.

The root tree shown in Figure 23 (left side) gives an overview of the properties created, while Figure 23 (right side) expands the Data Property “message”, showing that this kind of data is used by the UI Elements “Message Box”, “Notification”, “Tool Tip” and “Modal Window”. “Message” has also been defined to range the primitive data String. Table 6 shows the whole set of Data Properties created, their respective Domain Classes as well as their Datatypes. As some UI Elements can handle another UI Elements or even different Datatypes, we have defined the generic type “element” for modeling this property. For example, Menus present options for users, but these options can be of any type, i.e. images, text, or even another UI Element such as a Menu Item. Finally, notice that the only Data Property that does not use a Datatype is the property “Level”, which refers to the level of a Prototype.

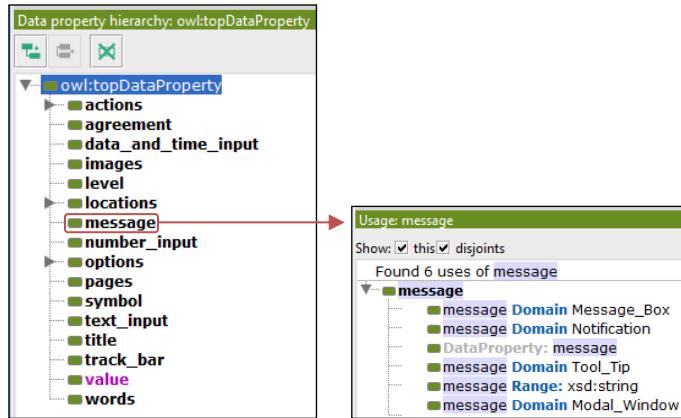


Figure 28. Left: Data Properties. Right: Data Property “message”.

Data Property	Domain Classes	Datatype
Actions	Menu Item, Link, Message Box, Button, Modal Window	xsd:element
State	-	xsd:boolean
Agreement	Notification	xsd:string
Data and Time Input	Calendar	xsd:dateTime
Images	Image Carousel	xsd:hexBinary
Level	Prototype	-
Locations	Breadcrumb	xsd:string
State	-	xsd:boolean
Message	Message Box, Notification, Text, Tool Tip, Modal Window	xsd:string
Number Input	Numeric Stepper	xsd:double
Options	Tabs Bar, Checkbox, Dropdown List, Toggle, List Box, Radio Button, Accordion, Menu, Progress Bar, Dropdown Button	xsd:element
State	-	xsd:boolean
Pages	Pagination	xsd:integer
Symbol	Icon	xsd:hexBinary
Text Input	Search Field, Text Field, Autocomplete	xsd:string
Title	Button, Field Set, Link, Label, Menu Item	xsd:string
Value	Slider	xsd:double xsd:string
Words	Tag	xsd:string

Table 6. Data Properties in the ontology.

4.2.4 Platform Concepts

Concepts of supported platforms are modeled in the ontology to determine which kind of UI is supported by the model and how its interactive elements will behave for each implementation. Having different presentations and behaviors depending on the platform they are implemented; the modeling of interactive elements must consider such particularities. The set of UI Elements that suits each platform is presented as Object Properties in the subsection 4.2.2.

So far, the ontology supports only interactive behaviors for web and mobile UIs. As shown in Figure 24, the classes Web and Mobile have been modeled as specializations of the class Platform, which allows us to eventually cover other platforms in the future. As a consequence of such choice, only UI Elements that are supported by web and mobile environments have been

described in the superclass Interaction Elements. The example below illustrates distinct implementations of an interactive element “Calendar” for both web and mobile environments. Notice that even carrying the same semantics in both platforms, the way a user is supposed to interact with this component may differ in each platform. While in a web environment the user can directly select a day of a month by clicking backward and forward on the month/year selection buttons, in a mobile environment the user could be asked to interact with a calendar by scrolling the month, the day and the year separately.

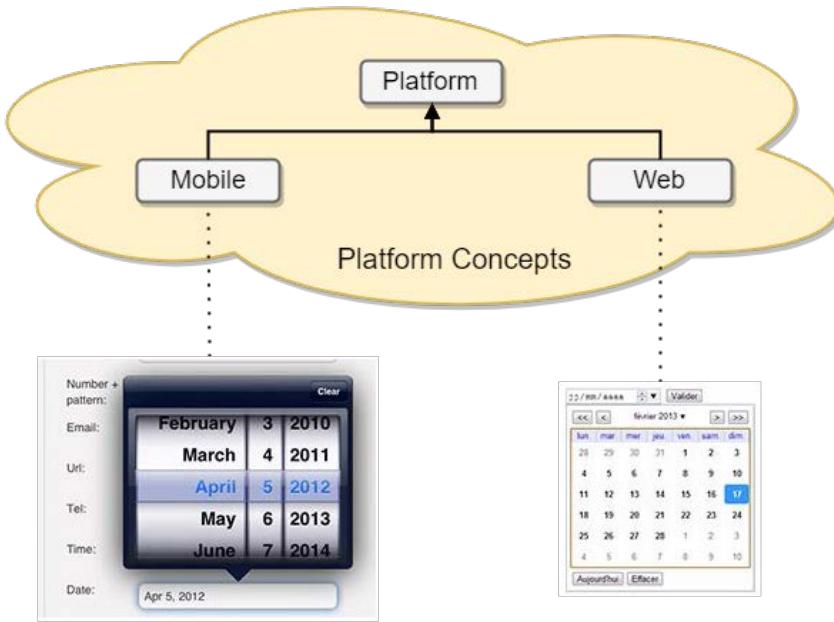


Figure 24. Example of Web and Mobile implementations of a Calendar.

4.2.5 UI Elements Concepts

UI Elements in the ontology represent an abstraction of GUI components in web and mobile platforms. Figure 25 illustrates a hierarchy of UI Elements. As we shall see, the four main superclasses are Container, Information Component, Input Control and Navigational Component. The first one contains elements that group other elements in a User Interface, such as Windows and Field Sets. The second one contains elements in charge of displaying information to the users such as Labels and Message Boxes. The third one represents elements in which users provide inputs to the system such as Buttons and Text Fields. Finally, the last one contains elements useful to navigate through the system such as Links and Menus. Some elements like Dialog Windows, for example, are inherited by more than one superclass, once they keep semantic characteristics of Containers and Information Components as well.

The complete list of UI Elements modeled in the ontology is presented in Table 7, specifying for each one: the correspondent superclass, a brief description and both Data and Object Properties associated. In Data Properties (DP) is identified the type of data handled by the UI Element itself. In Object Properties (OP) is identified whether the UI Elements are supported by web (OP: *usesAsA WebElement*) and/or mobile (OP: *usesAsAMobileElement*) platforms. It is also identified whether some UI Element has an Object Property *allowsUnique* or *allowsMultiple* associated to its Data Properties.

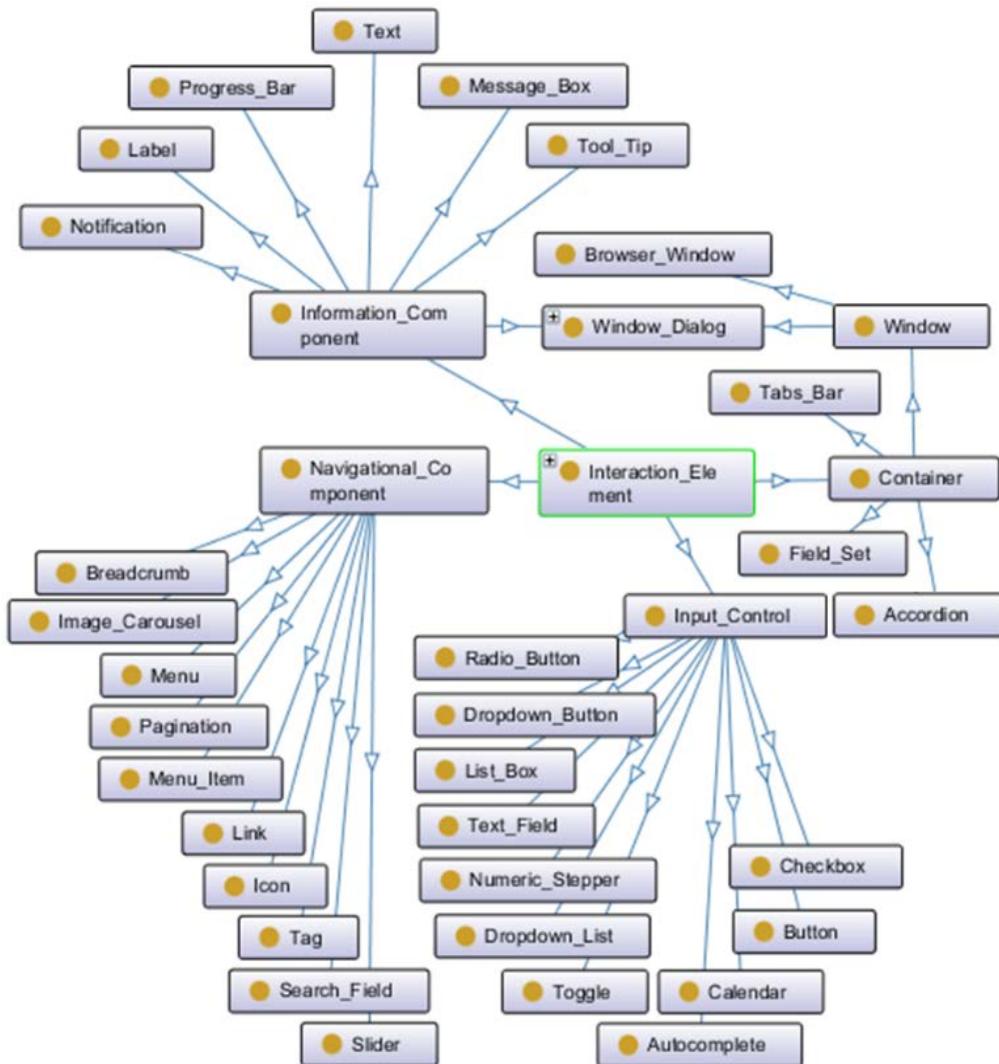


Figure 25. Cloud of User Interface (UI) Elements.

	Int. Element	Description	Properties
Container	Accordion	An Accordion is a vertically stacked list of items that utilizes show/hide functionality. When a label is clicked, it expands the section showing the content within. There can have one or more items showing at a time and may have default states that reveal one or more sections without the user clicking.	DP: options OP (usesAsA): <i>WebElement, MobileElement</i>
	Field Set	A Field Set element represents a set of form controls optionally grouped under a common name.	DP: title OP (usesAsA): <i>WebElement, MobileElement</i>
	Tabs Bar	A Tab Bar is a container widget that has typically multiple Tab Bar Buttons, which controls visibility of views. It can be used as a tab container.	DP: options OP (usesAsA): <i>WebElement, MobileElement</i>
	Window	A Window is an area on the screen that displays information, with its contents being displayed independently from the rest of the screen.	-

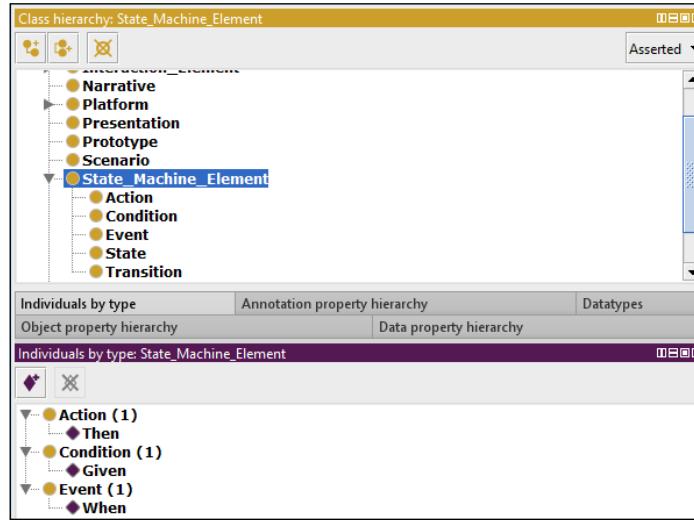
Window	Browser Window	The top of a typical Web browser window contains a title bar that displays the title of the current page. Below the title is a toolbar with back and forward buttons, an address field, bookmarks, and other navigation buttons. Below the toolbar is the content of the current Web page. The bottom of the window may contain a status bar that displays the page loading status.	OP (usesAsA): <i>WebElement</i>
	Window Dialog	A Window or Dialog Box is a small window that communicates information to the user and prompts them for a response.	OP (usesAsA): <i>WebElement</i>
Window Dialog	Modal Window	A Modal Window requires users to interact with it in some way before they can return to the system.	DP: actions, message OP (usesAsA): <i>WebElement</i>
Information Component	Label	A Label displays content classification.	DP: title OP (usesAsA): <i>WebElement, MobileElement</i>
	Message Box	A Message Box is a small window that provides information to users and requires them to take an action before they can move forward.	DP: actions, message OP (usesAsA): <i>WebElement, MobileElement</i>
	Notification	A Notification is an update message that announces something new for the user to see. Notifications are typically used to indicate items such as, the successful completion of a task, or an error or warning message.	DP: agreement, message OP (usesAsA): <i>WebElement, MobileElement</i>
	Progress Bar	A Progress Bar indicates where a user is as they advance through a series of steps in a process. Typically, progress bars are not clickable.	DP: options OP (usesAsA): <i>WebElement, MobileElement</i>
	Text	Informative content in a page.	DP: message OP (usesAsA): <i>WebElement, MobileElement</i>
	Tool Tip	A Tooltip allows a user to see hints when they hover over an item indicating the name or purpose of the item.	DP: message OP (usesAsA): <i>WebElement, MobileElement</i>
	Window Dialog	-	-
	Autocomplete	The Autocomplete widget provides suggestions while you type into the field.	DP: text_input OP (usesAsA): <i>WebElement</i>
Input Control	Button	A Button indicates an action upon touch and is typically labeled using text, an icon, or both.	DP: actions, title OP (usesAsA): <i>WebElement, MobileElement</i>

	Calendar	A Calendar (date picker) allows users to select a date and/or time. By using the picker, the information is consistently formatted and input into the system.	DP: data_and_time_input OP (usesAsA): WebElement, MobileElement
	Checkbox	Checkboxes allow the user to select one or more options from a set. It is usually best to present checkboxes in a vertical list. More than one column is acceptable as well if the list is long enough that it might require scrolling or if comparison of terms might be necessary.	DP: options OP (usesAsA): WebElement, MobileElement OP: allowsMultiple
	Dropdown Button	The Dropdown Button consists of a button that when clicked displays a drop-down list of mutually exclusive items.	DP: options OP (usesAsA): WebElement, MobileElement OP: allowsUnique
	Dropdown List	Dropdown Lists allow users to select one item at a time, similarly to radio buttons, but are more compact allowing you to save space. Consider adding text to the field, such as ‘Select one’ to help the user recognize the necessary action.	DP: options OP (usesAsA): WebElement, MobileElement OP: allowsUnique
	List Box	List Boxes, like Checkboxes, allow users to select multiple items at a time, but are more compact and can support a longer list of options if needed.	DP: options OP (usesAsA): WebElement, MobileElement OP: allowsMultiple
	Numeric Stepper	A Numeric Stepper serves the same function as a Numeric Input Object. It is a method of entering numeric data in which the numbers can be typed directly into the input object. However, numeric values can also be adjusted by using up and down arrows next to the numeric input. Clicking the up and down arrows normally causes the value to increment by one.	DP: number_input OP (usesAsA): WebElement, MobileElement
	Radio Button	Radio Buttons are used to allow users to select one item at a time.	DP: options OP (usesAsA): WebElement, MobileElement OP: allowsUnique
	Text Field	Text Fields allow users to enter text. It can allow either a single line or multiple lines of text.	DP: text_input OP (usesAsA): WebElement, MobileElement
	Toggle	A Toggle button allows the user to change a setting between two states. They are most effective when the on/off states are visually distinct.	DP: options OP (usesAsA): WebElement, MobileElement

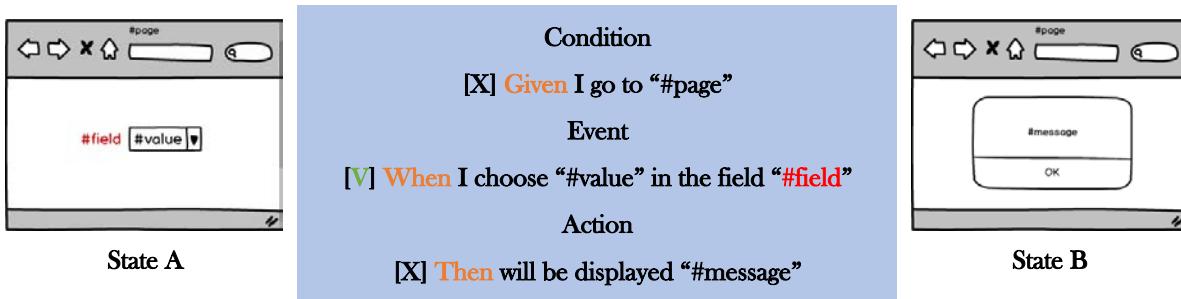
Navigational Component	Grid	A Grid or a Datagrid is a graphical control element that presents a tabular view of data.	OP: <i>allowsUnique</i> DP: <i>text_input</i> OP (usesAsA): <i>WebElement, MobileElement</i>
	Breadcrumb	Breadcrumbs allow users to identify their current location within the system by providing a clickable trail of proceeding pages to navigate.	DP: <i>locations</i> OP (usesAsA): <i>WebElement</i>
	Icon	An Icon is a simplified image serving as an intuitive symbol that is used to help users to navigate the system. Typically, icons are hyperlinked.	DP: <i>symbol</i> OP (usesAsA): <i>WebElement, MobileElement</i>
	Image Carousel	Image Carousels allow users to browse through a set of items and make a selection of one if they so choose. Typically, the images are hyperlinked.	DP: <i>images</i> OP (usesAsA): <i>WebElement</i>
	Link	A Link is a reference to data that can be directly followed by clicking. It points to a whole document or to a specific element within a document.	DP: <i>actions, title</i> OP (usesAsA): <i>WebElement</i>
	Menu	Menu is a list of options or commands presented to an operator.	DP: <i>options</i> OP (usesAsA): <i>WebElement, MobileElement</i>
	Menu Item	A Menu Item is a resultant item in a list of options or commands presented to an operator by clicking in a menu.	DP: <i>actions, title</i> OP (usesAsA): <i>WebElement, MobileElement</i>
	Pagination	Pagination divides content up between pages and allows users to skip between pages or go in order through the content.	DP: <i>pages</i> OP (usesAsA): <i>WebElement</i>
	Search Field	A search box allows users to enter a keyword or phrase (query) and submit it to search the index with the intention of getting back the most relevant results. Typically, search fields are single-line text boxes and are often accompanied by a search button.	DP: <i>text_input</i> OP (usesAsA): <i>WebElement, MobileElement</i>
	Slider	A slider, also known as a track bar, allows users to set or adjust a value. When the user changes the value, it does not change the format of the interface or other info on the screen.	DP: <i>value</i> OP (usesAsA): <i>WebElement, MobileElement</i>
	Tag	Tags allow users to find content in the same category. Some tagging systems also allow users to apply their own tags to content by entering them into the system.	DP: <i>words</i> OP (usesAsA): <i>WebElement</i>
	Tree	With a Tree, we can display hierarchical data. Each row displayed by the Tree contains exactly one item of data, which is called a node. Every Tree has a root node from which all nodes descend. By default, the Tree displays the root node. A node can either have children or not. We refer to nodes that can have children — whether or not they currently have children — as branch nodes. Nodes that cannot have children are leaf nodes.	DP: <i>actions</i> OP (usesAsA): <i>WebElement</i>

Table 7. UI Elements in the ontology.

4.2.6 State Machine Concepts

**Figure 26.** State Machine Elements and their Individuals.

The dialog part of a User Interface, as illustrated by Figure 26, is described in the ontology using concepts borrowed from abstract State Machines. A Scenario meant to be run in a given UI is represented as a Transition, illustrated by Figure 27. States are used to represent the original and resulting UIs after a transition occur (States A and B in Figure 27). Scenarios in the Transition state always have at least one or more Conditions (represented in Scenarios by the “Given” clause), one or more Events (represented in Scenarios by the “When” clause), and one or more Actions (represented in Scenarios by the “Then” clause). These constraints have been guaranteed in our tools which implement these ontological concepts. The clauses “Given”, “When” and “Then” have been modeled as Individuals of each respective class.

**Figure 27.** A Transition being represented in the State Machine.

4.2.7 Scenario-Based Concepts

Scenario-based concepts allow us modeling behaviors that describe how users are supposed to interact with the systems whilst manipulating graphical elements of the User Interface. An example of behavior specification is illustrated by Figure 28.

Behaviors are structured and described in natural language, so that they can also be read by humans. The specification of behaviors encompasses when the interaction can be performed (using “Given”, “When” and/or “Then” clauses – which are Individuals in the ontology), and which graphical elements (i.e. Radio Button, CheckBox, Calendar, Link, etc. – which are classes

in the ontology) can be affected. Altogether, behaviors and graphical elements are used to implement the test of expected system behavior. In the example in Figure 28, the behavior receives two parameters: a “\$elementName” and a “\$locatorParameters”. The first parameter is associated to data, the second parameter refers to the Interaction Element supported by this behavior: “RadioButton”, “CheckBox”, “Calendar” and “Link”. To comply with semantic rules, the behavior “*I choose |”\$elementName| referring to |”\$locatorParameters|” shown in Figure 28 can be modelled into a predefined behavior “chooseReferringTo” as shown in Figure 29.*

In the ontology, behaviors are modeled as Object Properties (OP). The ontology includes a large set of predefined behaviors grouped by context of use, as shown in Table 8. Notice that each Behavior is associated to diverse transition components (Context, Event and/or Action) that compose a Transition. The column UI Elements enlists the set of Interaction Elements that can fit to trigger a particular behavior.

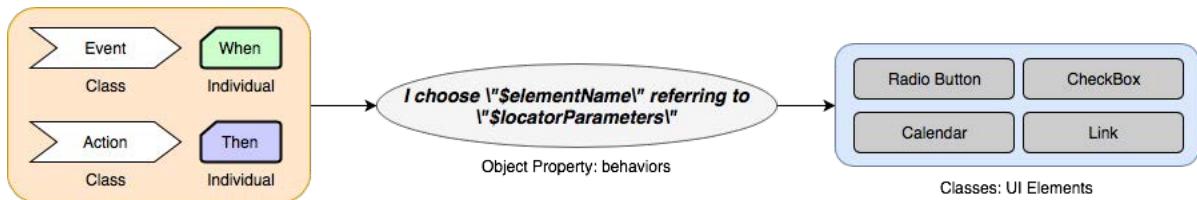


Figure 28. Components on the ontology used to specify a behavior.



Figure 29. Behavior “chooseReferringTo”.

Checkbox and Radio Button Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
theFieldIsUnchecked				Checkbox Radio Button
theFieldIsChecked				Checkbox Radio Button
assureTheFieldIsUnchecked				Checkbox
assureTheFieldIsChecked				Checkbox
Common Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
choose ≡ select				Calendar Checkbox Radio Button Link

<i>chooseByIndexInTheField</i>				Dropdown List
<i>chooseReferringTo</i>				Calendar Checkbox Radio Button Link
<i>chooseTheOptionOfValueInTheField</i>				Dropdown List
<i>clickOn</i>				Menu Menu Item Button Link
<i>clickOnReferringTo</i>				Menu Menu Item Button Link Grid
<i>doNotTypeAnyValueToTheField</i> \equiv <i>resetTheValueOfTheField</i>				Text Field
<i>goTo</i>				Browser Window
<i>goToWithTheParameters</i>				Browser Window
<i>isDisplayed</i>				Browser Window
<i>setInTheField</i> \equiv <i>tryToSetInTheField</i>				Dropdown List Text Field Autocomplete Calendar
<i>setInTheFieldReferringTo</i>				Dropdown List Text Field
<i>typeAndChooseInTheField</i> \equiv <i>informAndChooseInTheField</i>				Autocomplete
<i>willBeDisplayed</i>				Text
<i>willNotBeDisplayed</i>				Text
<i>willBeDisplayedInTheFieldTheValue</i>				Element
<i>willNotBeDisplayedInTheFieldTheValue</i>				Element
<i>willBeDisplayedTheValueInTheFieldReferringTo</i>				Element
<i>willNotBeDisplayedTheValueInTheFieldReferringTo</i>				Element
<i>isNotVisible</i>				Element
<i>valueReferringToIsNotVisible</i>				Element
<i>waitTheFieldBeVisibleClickableAndEnable</i>				Element
<i>waitTheFieldReferringToBeVisibleClickableAndEnable</i>				Element
<i>theElementIsVisibleAndDisable</i>				Element
<i>theElementReferringToIsVisibleAndDisable</i>				Element
<i>setInTheFieldAndTriggerTheEvent</i>				Text Field
<i>clickOnTheRowOfTheTree</i>				Tree
Data Generation Behaviors				
Behavior	Transition			UI Elements
	C	E	A	

<i>informARandomNumberWithPrefixInTheField</i>				Text Field
<i>informARandomNumberInTheField</i>				Text Field
Data Provider Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
<i>inform</i>				Grid
<i>informTheField ≡ informTheFields</i>				Grid
<i>selectFromDataSet</i>				-
<i>informTheValueOfTheField</i>				Element
<i>informKeyWithTheValue ≡ defineTheVariableWithTheValue</i>				-
<i>obtainTheValueFromTheField</i>				Element
Debug Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
<i>printOnTheConsoleTheValueOfTheVariable</i>				-
Dialog Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
<i>confirmTheDialogBox</i>				Window Dialog
<i>cancelTheDialogBox</i>				Window Dialog
<i>informTheValueInTheDialogBox</i>				Window Dialog
<i>willBeDisplayedInTheDialogBox</i>				Window Dialog
Mouse Control Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
<i>moveTheMouseOver</i>				Menu MenuItem Button Link
Table Behaviors				
Behavior	Transition			UI Elements
	C	E	A	
<i>clickOnTheRowOfTheTableReferringTo</i>				Grid
<i>storeTheCellOfTheTableIn</i>				Grid
<i>storeTheColumnOfTheTableIn</i>				Grid
<i>compareTheTextOfTheTableCellWith</i>				Grid
<i>compareTheTextOfTheTableColumnWith</i>				Grid
<i>clickOnTheCellOfTheTable</i>				Grid
<i>clickOnTheColumnOfTheTable</i>				Grid
<i>chooseTheOptionInTheCellOfTheTable</i>				Grid
<i>chooseTheOptionInTheColumnOfTheTable</i>				Grid

<i>typeTheTextInTheCellOfTheTable</i>				Grid
<i>typeTheTextInTheColumnOfTheTable</i>				Grid

Table 8. Predefined Behaviors described in the ontology.

The vocabulary chosen to express each behavior emerged from Scenarios specified in our past projects. It outlines only one of the several possible vocabularies to represent the same user's behaviors and could be extended in the future by more representative phrases or expressions. Some synonyms concerning the user's goal have been also identified in order to increase the expressivity of the ontology. For example, the behavior `doNotTypeAnyValueToTheField` is considered equivalent to the behavior `resetTheValueOfTheField` as they perform or assert exactly the same action on the affected UI element, looking for the same output. Likewise, the behavior `setInTheField` is equivalent to the behavior `tryToSetInTheField` as they refer to the same action. However, `tryToSetInTheField` better expresses violation attempts in the business rules.

4.2.8 Consistency Checking

```

INFO 13:00:09 ----- Running Reasoner -----
INFO 13:00:09 Pre-computing inferences:
INFO 13:00:09 - class hierarchy
INFO 13:00:09 - object property hierarchy
INFO 13:00:09 - data property hierarchy
INFO 13:00:09 - class assertions
INFO 13:00:09 - object property assertions
INFO 13:00:09 - data property assertions
INFO 13:00:09 - same individuals
INFO 13:00:14 Ontologies processed in 4926 ms by null
INFO 13:00:14

INFO 13:01:01 ----- Running Reasoner -----
INFO 13:01:01 Pre-computing inferences:
INFO 13:01:01 - class hierarchy
INFO 13:01:01 - object property hierarchy
INFO 13:01:01 - data property hierarchy
INFO 13:01:01 - class assertions
INFO 13:01:01 - object property assertions
INFO 13:01:01 - data property assertions
INFO 13:01:01 - same individuals
INFO 13:01:01 Ontologies processed in 64 ms by Pellet
INFO 13:01:01

```

Figure 30. Results of ontology processing: HermiT (top) and Pellet (bottom).

Consistency checking was done using the reasoners FaCT++, ELK, HermiT and Pellet. FaCT++ started identifying no support for the datatypes `xsd:base64Binary` and `xsd:hexBinary` used to range images and symbols in the Data Properties. Those properties have been used to define domains for objects in the classes `Image` `Carousel` and `Icon`, respectively. ELK has failed by no support to Data Property Domains as well as Data and Object Property Ranges. HermiT and Pellet have succeeded processing the ontology respectively in 4926 and 64 milliseconds, as presented in Figure 30.

4.3. Contributions, Limitations and Perspectives

The ontology presented in this chapter describes behaviors that report Steps of Scenarios performing actions directly on the UI through Interaction Elements. Thus, the ontological model is domain-free, which means that it is not dependent of business characteristics that are described in the User Stories. Specific business behaviors must be specified only for the systems to which they refer, not affecting the whole ontology. Therefore, it is possible to reuse Steps in multiple testing Scenarios of other systems requiring such kinds of user's actions. It brings a limitation

once Scenarios must be specified in the user interaction level, writing Steps for each click, selection, typing, etc. A possible solution to avoid this level of detail would be to work with higher-level behaviors that are described by user's tasks. Nonetheless, user's tasks often contain information from specific application domains. For example, high-level Steps like "*When I search for flights to 'Destination'*" encapsulate all low-level behaviors referring to individual clicks, selections, etc.; however, it also contains information that refers to the airline domain (i.e. behavior "search for flights"). Therefore, that Step would only make sense on that particular application domain. For further researches, it could be interesting to investigate domain ontologies to be used in parallel with our ontology, defining a higher-level business vocabulary database in which business behaviors could be mapped to a set of interactive behaviors, covering recurrent Scenarios for a specific domain, and avoiding them to be written every time a new interaction may be tested.

When representing the diverse Interaction Elements that can attend a given behavior, the ontology also allows extending multiple design solutions for the UI, representing exactly the same requirement in different perspectives. Besides modeling several concepts of the target artifacts, the ontology covers more than 60 interactive behaviors and almost 40 Interaction Elements for both web and mobile user interfaces. Thus, even if a Dropdown List has been chosen to attend for example a behavior `setInTheField` in a Prototype, an Auto Complete field could be chosen to attend this behavior on the Final UI, once both UI elements share the same ontological property for this behavior under testing. This kind of flexibility makes tests pass, leaving the designer free for choosing the best solutions in a given time of the project, without modifying the behavior specified for the system.

Another aspect to consider is that even having mapped synonyms for some specific behaviors, our approach does not provide any kind of semantic interpretation, i.e. the Steps might be specified exactly as they were defined on the ontology. The JBehave plugin for Eclipse shows (through different colors) if the Step being written exists or not on the ontology. This resource reduces the workload to remember as exactly some behavior has been described on the ontology and will be presented in chapter 6. On one hand, the restricted vocabulary seems to bring less flexibility to designers, testers and requirements engineers. Nonetheless, on the other hand, it establishes a common vocabulary, avoiding typical problems of ambiguity and incompleteness in requirements and testing specifications. Further studies on Natural Language Processing (NLP) techniques might help to improve the process of specification adding more flexibility to write Scenarios that could be semantically interpreted to meet the behaviors described on the ontology. This issue is certainly a worthwhile topic for further research.

It is also worthy of mention that the concepts and definitions in the ontology presented herein include one of the possible solutions for addressing and describing behaviors and their relations with UIs. Despite the fact that our ontology covers concepts available in well-known languages such as MBUI, UsiXML and SCXML, we do not assume that the coverage is exhaustive. In principle, the adequacy of a given set of elements present in the ontology to the system or project under development is our modeling stopping criterion. We envision that other behaviors, concepts and relationships might be included in the future to express idiosyncrasies of specific interaction techniques (ex. multimodal interaction techniques) and/or specific platforms (ex. ambient systems), or even to increase the coverage of Interaction Elements due to the emergence of new elements for web and mobile platforms. To do so, new elements can be added by direct imports into the ontology or simply by adding new more expressive behaviors to the Object Property "behaviors" and linking them to the appropriate set of Interaction Elements.

Finally, this ontology has been developed primarily to support the assessment of GUIs. Nevertheless, along this thesis, we will explore the use of the ontology to also support the assessment of different artifacts that compose the design of a User Interface. As the ontology has been designed in a behavior-based way and supported by a state machine, only scenario-based artifacts, i.e. artifacts that use scenarios to perform and/or simulate user activities in the system, are supported for testing purposes. This characteristic will be explored in the next chapters.

4.4. Resultant Publications

Silva, T. R., Hak, J.-L. & Winckler, M. (2017). A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces. International Journal of Semantic Computing, 11 (04), pp. 513-539. DOI: <http://doi.org/10.1142/S1793351X17400219>. (Silva, Hak and Winckler, 2017b)

Silva, T. R., Hak, J. L. & Winckler, M. (2017). A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems. In: 2017 IEEE 11th International Conference on Semantic Computing (ICSC 2017), pp. 250-257. IEEE. DOI: <http://doi.org/10.1109/ICSC.2017.73>. (Silva, Hak and Winckler, 2017a)

Chapter 5

Modeling and Assessing Task Models

Summary

This chapter details our strategy for modeling and assessing task models following our approach presented in chapter 3. The chapter begins by presenting the HAMSTERS notation which will be used for modeling and assessing our task models. The chapter continues by resuming the case study proposed previously, with task models being used to design user's tasks. In the sequence, we present firstly an orderly strategy for getting task models already consistent with the set of user requirements specified by users. The example of tasks we explore is an excerpt of the searching flight activity already modeled in a high level of abstraction in the BPMN model presented in chapter 3.

In the second section, we explore our strategy for assessing the resultant task models. This section is presented in 3 steps. The first one refers to the extraction of possible scenarios from a designed task model, formatting them to meet the ontological pattern. The second one refers to the process of mapping elements from the task model for checking whether they are consistent with the respective elements in the User Stories, and hence with the ontology. The third and last step presents how our strategy has been implemented to support testing in an automated way. Lastly, we present a discussion concerning the challenges of assessing task models and the limits of the approach.

As discussed in chapter 2, task models can serve many purposes, from modeling users' activities in early phases of development, until supporting test case generation in later phases of development (Campos *et al.*, 2017). They can evolve along the different phases of development or be thrown away as soon as user requirements have been settled up, and a consistent design of the user interface has been concluded. In this chapter, we adopt a use of task models serving as an early and evolutive design artifact for modeling aspects of functional user requirements.

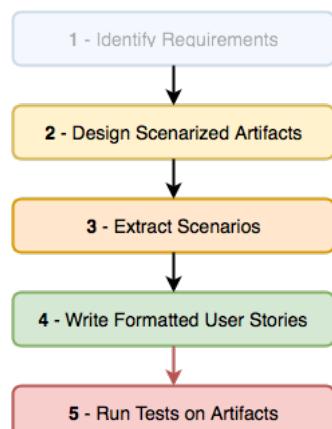


Figure 31. One of the alternatives to perform our approach.

Our strategy concerning the assessment of task models consists in checking their consistency with respect to a previously-defined requirements specification. As highlighted in chapter 3 (section 3.2.3), there are alternatives for performing our approach. As such, task models can be designed from the beginning for matching the requirements specification or, if they have already been designed, for supporting the development of the requirements specification which will benefit from a preliminary analysis of user's tasks. One of these alternatives is reproduced in Figure 31. It will be used in this chapter to present the approach. Following this alternative, we had already identified the requirements to be modeled (1) in chapter 3 (section 3.3.1). Thus, in this chapter, we will in the sequence: design the task models (as part of the scenarized artifacts) (2), extract scenarios from them

(3), write our formatted User Stories based on such extracted scenarios (4), and finally run tests on the artifacts (5).

Task models can be designed through a diverse set of notations and tools. For being assessed under our approach though, they need to comply with two premises:

- Allow extraction of scenarios by running the model.
- Export source files of both reference model and extracted scenarios in a markup language.

Although in theory task models in any notation may be assessed since they comply with these two premises, our implementation should be adapted to understand the formalism used by such notations to describe the task model and the scenarios extracted from them. Our strategy for testing performs a static assessment of the source files by means of a syntactic and semantic analyzes of the target source files. An advantage of this approach is that, unlike co-execution approaches where both artifacts under testing must be prepared for assessment by annotating or modifying their source files, with our approach we have no need to intervene in the source code of the target artifacts, i.e. artifacts do not need to be prepared for testing by designers, so task models and requirements specifications can be assessed in their original state.

For the demonstration we propose in this chapter, we make use of task models modeled by HAMSTERS once the notation and tool fit our two premises stated above. HAMSTERS exports its reference models and extracted scenarios using the XML standard, a well-adopted markup language, so recognized by our approach. The task modeling and the extraction of scenarios that will be presented hereafter has been made by using the HAMSTERS tool, whilst the implementation of the assessment has been made by using the respective XML source files produced by the HAMSTERS tool for each model. The next section presents a brief overview of the HAMSTERS' notation and tool support, and the following sections present our strategy for modeling and testing based on the alternative for running described in Figure 31.

5.1. An Overview of HAMSTERS

Human-centered Assessment and Modeling to Support Task Engineering for Resilient Systems (HAMSTERS) (Martinie, Palanque and Winckler, 2011) is a notation inspired by other existing ones for task modeling, especially CTT (Paternò, 2003), and, according to the authors, has been designed to remain compatible with it (from the point of view of people building the models) as models are hierarchical and are graphically represented featuring operators between the tasks. However, HAMSTERS includes extensions such as pre-conditions associated with task executions, data flow across task models, and more detailed interactive tasks. HAMSTERS' models can be edited and simulated in a dedicated environment which also provides a dedicated API for observing, editing, and simulating events, making it possible to connect task models to system models (Navarre *et al.*, 2001; Barboni *et al.*, 2010). HAMSTERS has been introduced in 2011 and several versions have been released since them. In this thesis, we are adopting the version 4.0 of HAMSTERS. Thus, the components we present hereafter, and which be used along the case studies are based on existing elements until such a version.

5.1.1. Task Types

Table 9 illustrates some of the HAMSTERS' constructs that are required for structuring models, including:

- Abstract task is a task that involves sub-tasks of different types.
- System task is a task performed only by the system.
- User task is a generic task describing a user activity. It can be specialized as a Motor task (e.g. a physical activity), a Cognitive task (e.g. decision making, analysis), or Perceptive task (e.g. perception of alert).
- Interactive task represents an interaction between the user and the system; it can be refined into Input task when the users provide input to the system, Output task when the system provides an output to the user and Input/Output task which is a mix of both but performed in an atomic way.

Task type	Icons in HAMSTERS task model			
Abstract Task	 Abstract task			
System Task	 System task			
User Tasks	 User task			
Interactive Tasks	 Interactive task			
	 Inputtask			
	 Output task			
	 InputOutput task			

Table 9. Task types in HAMSTERS.

Tasks can also have properties. Tasks may be optional, iterative or both optional and iterative. The representation of these properties is depicted in Figure 32 below. In addition, minimum and maximum execution time can also be set for tasks, and particularly for iterative tasks, it can also be set the number of iterations they support.

**Figure 32.** Example of Task Properties.

The notation also provides a composition mechanism to describe sub-routines. A sub-routine is a group of activities that a user performs several times, possibly in different contexts which might exhibit different types of information flows. The sub-routine is then modeled in a dedicated model where the root task is the icon of that sub-routine. A sub-routine contains:

- The name of the sub-routine.
- The icon of an “Abstract” task type (as the sub-routine consists of a group of tasks that can belong to different types).
- Specialized input and output ports attached both to the left side and to the right side of the icon. The graphical symbol of these specialized ports can be filled (if they handle parameters) or not (if they do not). These ports are mechanisms for representing required parameters to and/or from sub-routines, thus providing explicit representation of data flow during task execution.

5.1.2. Operators

Additionally, temporal relationships between tasks are represented by means of operators. The operator “Enable” (\gg) describes that the tasks T1 and T2 occur sequentially, one after the other. The operator “Concurrent” ($| |$) describes that the tasks T1 and T2 can be performed simultaneously. The operator “Choice” ($\{ \}$) describes the user performing the tasks T1 or T2, but the choice of one implies that the other will be disabled. The operator “Disable” ($[>]$) describes that the starting of the task T1 leads to a definitive interruption of the task T2. The operator “Suspend-resume” ($| >$) describes that the starting of the task T1 leads to a temporary interruption of the task T2; T1 can be restarted at any time and then be interrupted again by the task T2, while T1 is not complete. Finally, the operator “Order independent” ($| = |$) describes that the user can choose whether he will perform the tasks T1 or T2 first. This operator also indicates that the task selected to be executed first will be completed before moving to the next. Table 10 summarizes such operators.

It is the use of these operators to link tasks in the model that allows extracting of the possible scenarios to be performed in the system. This is done by following the multiple achievable paths in the model, with each combination of them generating an executable scenario.

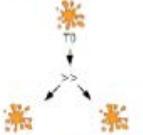
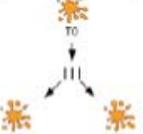
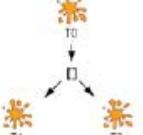
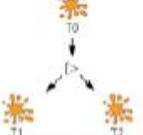
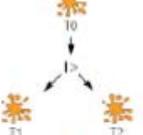
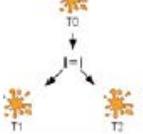
Temporal operator in a task model	Operator type	Description
	Enable	In order to accomplish T0, T2 is executed after T1.
	Concurrent	In order to accomplish T0, T1 and T2 are executed at the same time.
	Choice	In order to accomplish T0, T1 is executed OR T2 is executed
	Disable	In order to accomplish T0, execution of T2 interrupts the execution of T1
	Suspend - resume	In order to accomplish T0, execution of T2 interrupts the execution of T1, T1 execution is resumed after T2.
	Order independent	In order to accomplish T0, T1 is executed then T2 OR T2 is executed then T1

Table 10. Illustration of the operator types within HAMSTERS.

5.1.3. Extracting Scenarios

HAMSTERS tool allows models to be executed through the simulation mode (illustrated in Figure 33). By using the it, we can view the current tasks that are available for execution (list in the upper part of the simulation panel in Figure 33), and the scenario, i.e. the tasks that have been executed (list in the lower part of the simulation panel in Figure 33). Additionally, the tasks which are available for execution are highlighted in green in the task model (in the central part in Figure 33). By extracting all the possible scenarios that could be performed in the model, we have a big picture about everything (in terms of tasks) that can be done with the system.

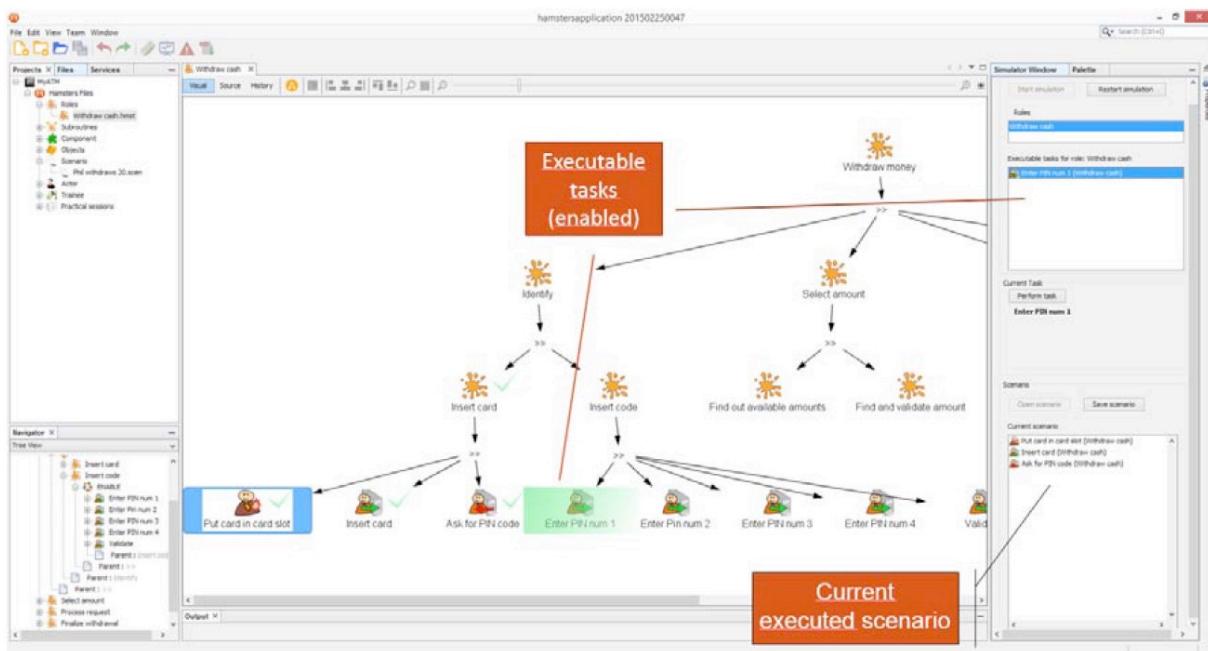


Figure 33. Representation of executable and executed tasks during simulation.

5.1.4. Handling Data

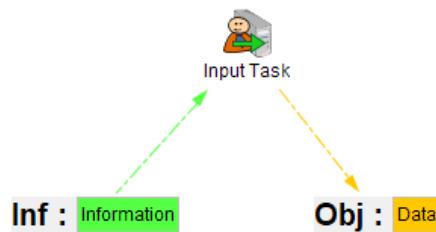


Figure 34. Example of “Information” and “Data” handling.

HAMSTERS expressive power goes beyond most other task modeling notations particularly by providing detailed means for describing data that is required and manipulated (Martinie *et al.*, 2013) in order to accomplish tasks. Information (“Inf:” followed by a text box) may be required for execution of a system task, but it also may be required by the user to accomplish a task. Objects (“Obj:” followed by a text box), on the other hand, are used for indicating that some data will be provided when performing an input task by the user. These elements are exemplified in Figure 34, where the user considers a given information for performing and input task (arrow from the information to the input task) and then, when performing such task, he/she uses such information as an actual data that will be provided for the system (arrow from the input task to

the object). By using the HAMSTERS' simulation mode, we can set test data on runtime when performing an input task that points to an object in the model.

5.2. Modeling User's Tasks

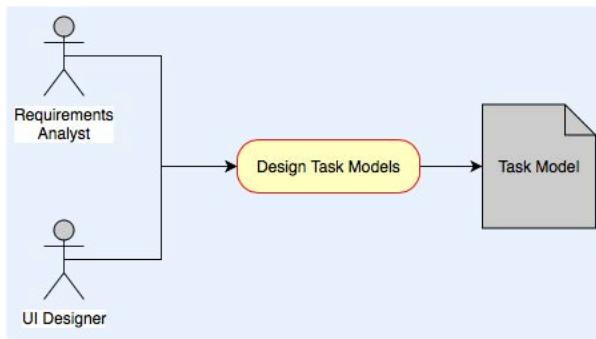


Figure 35. Activity of creating task models.

The task models presented hereafter have been modeled using the HAMSTERS notation and are based on the BPMN model designed in chapter 3 (section 3.3). The activity of modeling user's tasks described in this section corresponds to "Design Task Models" in our micro-process presented in chapter 3 (Figure 35). This activity is performed in collaboration between Requirements Analysts and UI Designers.

By resuming the illustrative case study started in chapter 3, we assume a generic workflow for flight reservations, not following any specific business process of a given company. The tasks represented below are focused on the processes of searching and choosing flights presented in the BPMN model.

Figure 36 presents respectively the extract of the business process selected for modeling and the resultant task models. In the transition (a), the initial business activity "Search Flights" has been mapped to the abstract/iterative task "Search Flights" once it is performed by the user. This task is refined in an ordered sequence of input/output tasks (operator "enable"). First, the user goes to the web page where he provides data for search (input task "Go to Find Flights"). Next, the user effectively provides a set of data for searching his flights (abstract task "Provide Data"), submits the search (input task "Submit Search"), and finally verifies the resultant list of flights (abstract task "Verify List of Flights"). These are sequential user tasks (operator "Enable"). For the abstract task "Verify List of Flights", the system actually provides the list of available flights (output task "Present List of Available Flights") and then the subtask "Choose Flights" becomes available to be performed by the user. It matches with the business activity "Verify List of Flights" in the BPMN model.

For providing the set of data for searching ("Inf:"), the user can inform in any other (operator "Order independent"): departure (abstract task "Inform Departure"), destination (abstract task "Inform Destination"), number of passengers (input task "Inform Number of Passengers"), departure date (input task "Set Departure Date"), and trip type (abstract task "Choose Trip Type"). Notice that the use of the operator "Order independent" allows the extraction of scenarios from this model with those tasks presented in any order.

The abstract tasks "Inform Departure" and "Inform Destination" originate a sequence of three tasks. The first one in which the user informs a departure (or arrival) city (respectively the input tasks "Inform Departure City" and "Inform Arrival City"). The second one in which the system provides a list of airports in the city (output task "Provide List of Airports"). Finally, the third one in which the user chooses the departure (or arrival) airport (respectively the input tasks "Choose Departure Airport" and "Choose Arrival Airport"). The abstract task "Choose Trip Type" is actually a decision task once the user can choose (operator "Choice") between a one-way (input task "Select One-way Trip") and a round trip (input task "Select Round Trip"). If he chooses a round trip, he needs to inform the arrival date (input task "Set Arrival Date") as well.

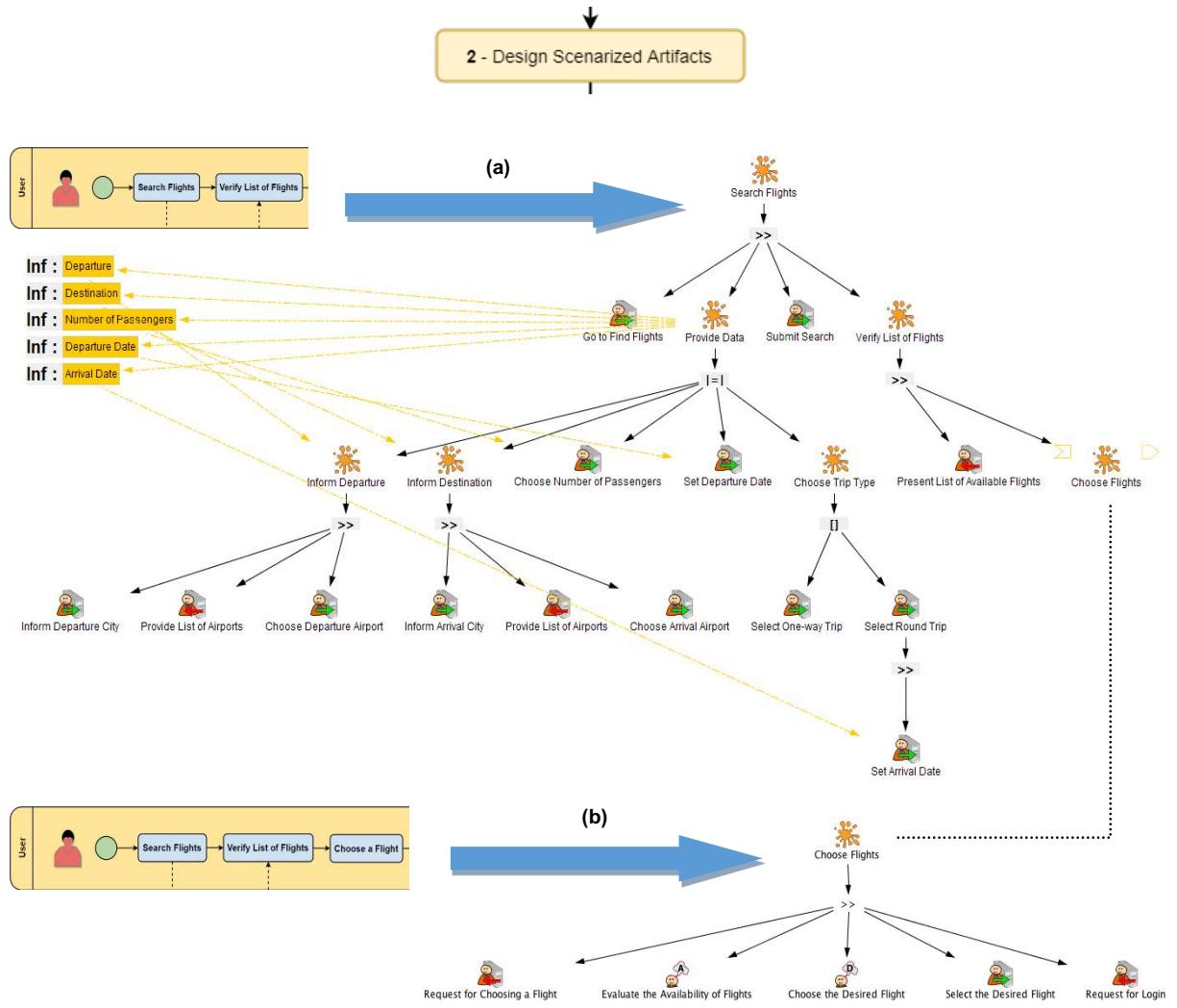


Figure 36. Mapping BPMN business activities to HAMSTERS user tasks.

In the transition (b) of Figure 36, we present the sequence of the flow. The business activity “Choose a Flight” has been mapped to the abstract/interactive task “Choose Flights” in the task model (notice that this same task has already been represented as the last abstract task in the first transition). Following the task “Choose Flights”, the system requests user for choosing a flight (output task “Request for Choosing a Flight”). Next, the user evaluates the availability of flights (cognitive analysis task “Evaluate the Availability of Flights”) and then makes a decision, choosing the desired flight (cognitive decision task “Choose the Desired Flight”). After the cognitive decision about which flight choose, the user finally performs the input task of selecting the desired flight (input task “Select the Desired Flight”). As a result, the system asks the user to provide his login information to proceed the booking with passengers and payment data (output task “Request for Login”).

Notice that business and task models are complementary. The business process model provides an overview of the activity flow of the system, emphasizing high-level processes involving diverse business actors. In a different way, the task model is more focused in describing detailed user tasks while interacting with the system, emphasizing lower level tasks. Thereby, task models provide more refined resources and descriptors to model user interactions than those provided by business process models.

5.3. Assessing User's Tasks

By following the alternative that we set up in the beginning of this chapter for performing the approach, the next activity for getting task models ready for testing is extracting scenarios from them. In the alternative we are following, such scenarios will serve as basis for formatting our previously specified User Stories in an attempt to get steps in User Stories and tasks in scenarios extracted from task models already consistent. After extracting scenarios from task models, and formatting the User Stories, we can run our tests on the task models.

5.3.1. Extracting Scenarios and Formatting User Stories

As task models are designed to support the multiple paths that users may accomplish to perform their tasks, assessing such models in a scenario-based approach involves initially extracting the possible scenarios that are supposed to be tested in a given interaction. It means that after modeling, designers should define which scenarios (or even all of them) from the model will be tested.

Based on the task model developed for the process of searching and choosing flights, we have used HAMSTERS to extract some possible scenarios that a user could perform in the system. HAMSTERS tool supports innately the extraction of scenarios from task models, by running them and extracting the possible achievable paths (3 - Extract Scenarios). Figure 38 illustrates an extraction result. The presented path simulates a scenario for a one-way trip. The ordered sequence of tasks for this scenario is listed at the top.

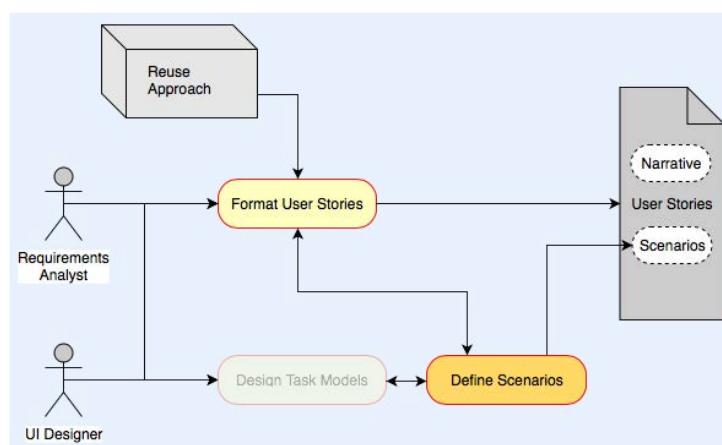


Figure 37. Activity of formatting User Stories.

The extracted scenario is then formatted to meet the User Story template (4 - Write formatted User Stories), with each ordered task being mapped to a testable common behavior described on the ontology presented in chapter 4. Thus, this mapping of common behaviors serves as a reuse approach for formatting the steps in the User Stories. The advantage of reusing such common behaviors is that they are already implemented for running tests on the target artifacts. The activity of formatting the User Stories

(illustrated in Figure 37 and exemplified below the Figure 38) is performed manually in collaboration between Requirements Analysts and UI Designers, so there is not any automatic transformation rule.

As an example, the illustrated scenario “One-Way Tickets Search” follows a possible path in the task model and describes the behavior for a one-way trip, using only data domains for testing. According to the business rule, the expected result for this search is a new screen presenting a “List of Available Flights”, in which the user might select the desired flight in a list of flights matching his search.

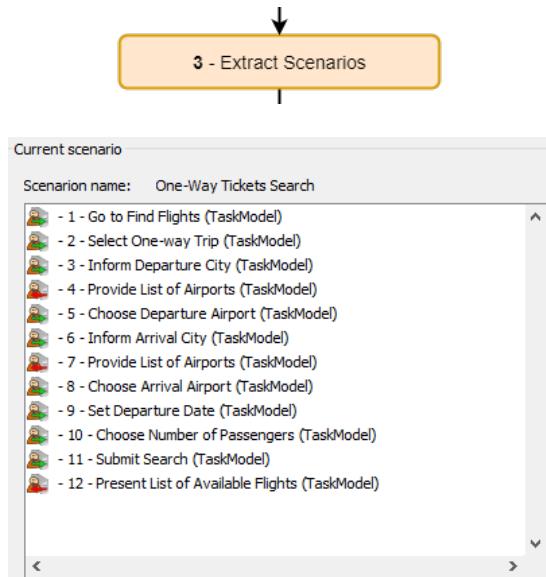
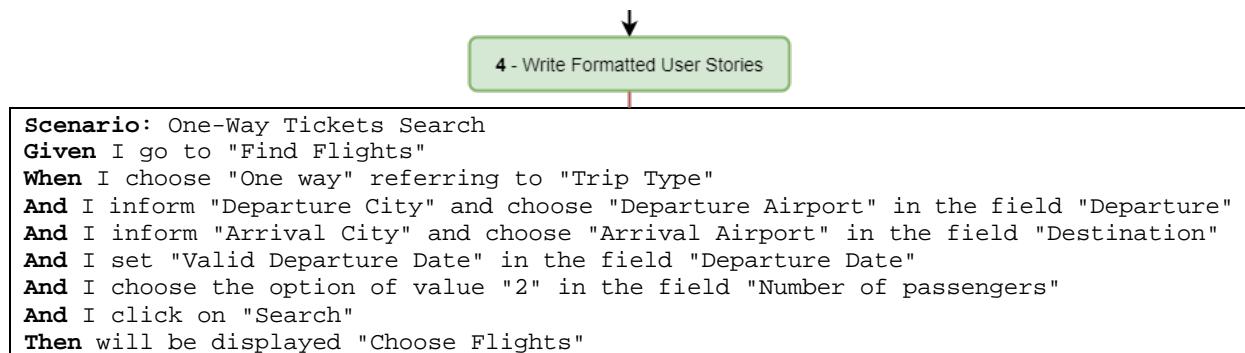


Figure 38. Scenarios being extracted from task models and then being formatted by the ontology as User Stories.



Exploring the set of possible scenarios that can be extracted from the task models we have designed in the previous section, we can establish a correlation between requirements identified in User Stories, their representation in terms of tasks and the extracted scenarios in both UCD and SE approaches, as stated in chapter 2. A possible solution for this correlation, considering two scenarios, and in accordance with the proposed ontology is presented in Table 11 and Table 12.

Requirement	Scenario	
	Extracted from Task Models (UCD approach)	Written in the BDD template (SE approach)
Narrative:  As a frequent traveller, I want to be able to search tickets, providing locations and dates, So that I can obtain information about rates and times of the flights.	Search Flights (abstract task)	Scenario: One-Way Tickets Search
	Go to Find Flights (input task)	Given I go to "Find flights"
	Select One-way Trip (input task)	When I choose "One way" referring to "Trip Type"
	Inform Departure (abstract task)	And I type "Paris" and choose "CDG - Paris Ch De Gaulle, France" in the field "Departure"
	Inform Destination (abstract task)	And I type "Dallas" and choose "DFW - Dallas Fort Worth International, TX" in the field "Destination"
	Set Departure Date (input task)	And I set "12/15/2016" in the field "Departure Date"

	Choose Number of Passengers (input task)	And I choose the option of value “2” in the field “Number of passengers”
	Submit Search (input task)	And I click on “Search”
	Present List of Available Flights (output task)	Then will be displayed “Choose Flights”
	Choose Flights (sub-routine)	-

Table 11. The correlation between requirements, tasks and scenarios in UCD and SE approaches for the User Story “Flight Tickets Search”.

Requirement	Scenario	
	Extracted from Task Models (UCD approach)	Written in the BDD template (SE approach)
Narrative: Travelers should be able to select available flights  As a frequent traveller, I want to get the list of flights and their rates and times, So that I can select the desired flight after a search of available flights.	Choose Flights (sub-routine)	Scenario: Select a diurnal flight
	Choose Flights (abstract task)	One-Way Tickets Search
	Request for Choosing a Flight (output task)	Given “Flights Page” is displayed
	Evaluate the Availability of Flights (cognitive analysis task)	-
	Choose the Desired Flight (cognitive decision task)	-
	Select the Desired Flight (input task)	When I click on “Flights” referring to “AA flight 6557, AA flight 51”
	Request for Login (output task)	Then “Optional log in” is displayed

Table 12. The correlation between requirements, tasks and scenarios in UCD and SE approaches for the User Story “Select the desired flight”.

Analyzing these correlations, we can make a set of important remarks. The first one is that the business value (such as defined in chapter 2 and represented in orange in the Narratives) and the testing component (represented in purple in the BDD scenario) allow us to implement test cases to validate the envisioned requirement, as well as checking when, after being implemented, this feature can be considered as “done” and correct (that correspond to the business value being achieved).

A second remark is that concerning the type of tasks mapped to scenarios in SE, as SE considers only tasks being performed by users when using an interactive system, User Stories in this context address only scenarios extracted from interactive tasks in task models. As highlighted in red in Table 12, cognitive tasks, for example, are not mapped to SE scenarios because they cannot be performed in the system.

Another remark is that the abstract tasks “Inform Departure” and “Inform Destination” highlighted in blue in Table 3 were detailed in the task model as a sequence of Input/Output interactive tasks. This happens because first the user informs a departure/destination city (Input task “Inform City”), then the system returns a list of airports in this city (Output task “Provide List of Airports”), and finally the user selects the desired airport (Input task “Select the Airport”). This behavior is typically represented by the interaction element AutoComplete in the UI design, in which the user types some text and the element dynamically returns a set of values that matches it. After that, the user is able to choose which value he wants. Because of that, this behavior was

represented with the step “...type and choose...” in the SE scenario, thus describing a double action in the UI.

A fourth point is that the sub-routine “Choose Flights” was represented in the first model (scenario: One-Way Tickets Search) as a result of the sequence of user tasks, and then detailed in the second model (scenario: Select a diurnal flight) as an abstract task. As the second scenario depends on the execution of the first one, the abstract task was represented in the SE scenario as a reference for the scenario “One-Way Tickets Search” that has just been performed. Thereby, the results of the scenario “One-Way Tickets Search” allow the choice of flights in the scenario “Select a diurnal flight”.

Finally, a last remark is that data are not directly modeled on task models. They should be informed during the extraction of scenarios. However, SE scenarios need these data to perform tests on the UI. Therefore, in the task modeling level, tasks are described in a generic way, as in the input task “Set Departure Date”, for example. When these tasks are extracted from the task models, in order to be testable, they need to receive an example of some representative data in that context (for example, the value “12/15/2016” as it has been done in the correspondent step “And I set ‘12/15/2016’ in the field ‘Depart’”). For testing purposes, when describing SE scenarios, it is crucial to design them with data that make the results succeed as well as with data that make the results fail. It is this mechanism that makes possible to bring a large and representative testing component for the requirements. These data can be provided for SE scenarios by multiple sources. They will be described in detail in the section 6.4.3 in chapter 6.

5.3.2. Elements Mapped for Testing

The equivalence of steps in User Stories and tasks in scenarios extracted from task models is assured by a formatting rule presented in Figure 37. Our testing algorithm (that will be presented in detail in the next section) performs such a rule in order to verify whether a behavior described in a step has an equivalent task to model it in the task model. The full mapping table considered by our algorithm is presented in the Appendix A of this thesis.

Step of Scenario	Task Name
When I set “Valid Departure Date” in the field “Depart Date”	Set Departure Date

Figure 39. Formatting rule for assessing steps and tasks.

This rule aims to eliminate unnecessary components of the step that do not need to be present in the task. The component “When” refers to the transition in the state machine which is not addressed in a task model. The subject “I” signalizes that is the user who performs the task. Tasks models encompass the definition of user role, so the statement “I” refers to any users that might correspond to the role assigned to the task model. The verb “set” indicates the action that will be performed by the user, so it begins naming the task in the task model. The value “Valid Departure Date” indicates a data domain that will be used to perform and test the task (information that is not present in the task name). The phrase complement “in the field” just signalizes that an interaction component (a “field”) will be called. Finally, the target field “Departure Date” indicates the name of the interaction component that will be affected by this task, so it composes the final name of the task in the task model. The Table 13 below summarizes the use of such components for mapping steps of scenarios and tasks. A complete concept mapping table for the tasks and behaviors supported by the ontology is presented in the Appendix A.

Component	Description	Use for naming tasks
When	Refers to the transition in the state machine.	Not used because it is not addressed in a task model.
I	Signalizes that is the user who performs the task.	Not used because the task models encompass the definition of user role.
set	Indicates the action that will be performed by the user.	Used for beginning the naming of the task in the task model.
“Valid Departure Date”	Indicates a data domain that will be used to perform and test the task.	Not used because such information is not present in the task name.
in the field	Signalizes that an interaction component (a “field”) will be called.	Not used because it is just a phrase complement.
“Departure Date”	Indicates the name of the interaction component that will be affected by this task.	Used for composing the final name of the task in the task model.

Table 13. Task name components construction.

The testing of UI design artifacts like task models is conducted by automatically checking whether user and business requirements have been consistently modeled. By way of example, Table 14 gives the correspondence of concepts in the task model, in the ontology, and in the step that would be performed by our algorithm when assessing the scenarios. Therein, the consistency of the requirements representation for the scenario “One-Way Tickets Search” is being checked in the respective task model.

Concepts		Step of Scenario
Task Model	Ontology	
Input Task: Go to Find Flights	Behavior: <i>goTo</i>	Given I go to “Find Flights”
Abstract Task: Choose Trip Type	Behavior: <i>chooseReferringTo</i>	When I choose “One way” referring to “Trip Type”
Abstract Task: Inform Departure	Behavior: <i>informAndChooseInTheField</i>	And I inform “Departure City” and choose “Departure Airport” in the field “Departure”
Abstract Task: Inform Destination	Behavior: <i>informAndChooseInTheField</i>	And I inform “Arrival City” and choose “Arrival Airport” in the field “Destination”
Input Task: Set Departure Date	Behavior: <i>setInTheField</i>	And I set “Valid Departure Date” in the field “Departure Date”
Input Task: Choose Number of Passengers	Behavior: <i>chooseTheOptionOfValueInTheField</i>	And I choose the option of value “2” in the field “Number of passengers”
Input Task: Submit Search	Behavior: <i>clickOn</i>	And I click on “Search”
Output Task: Present List of Available Flights	Behavior: <i>willBeDisplayed</i>	Then will be displayed “List of Available Flights”

Table 14. Concept mapping for the scenario “One-Way Tickets Search”.

5.3.3. Implementation

We have conducted automated consistency checking on task models by parsing their resultant XML source files from the extracted scenarios produced by the HAMSTERS tool. To do so, we have implemented an integrated algorithm in Java using JDOM and JUnit for parsing and testing User Stories against these artifacts. This section describes how it has been implemented.

5.3.3.1. Pre-formatting Source Files

The first step for assessing the set of scenarios extracted from task models is to preformat their XML files. As each task model notation and tool has its own way to implement and export scenarios and models, and there is no such a standard for that, each notation would demand a different preformatting to be tested by our approach. We have implemented a solution for HAMSTERS in its current version (v4.0), but we have designed a flexible and open architecture where other notations could benefit from our approach by just implementing a new preformatting java class in accordance with their own patterns to implement scenarios and models.

HAMSTERS tool exports scenarios with only a reference to the task ID and the object ID that compose the flow. As such, we have to prepare the files for testing. So, before starting the assessing, we edit each scenario XML file to add:

- The name of the task referenced by each task ID.
- The information about the optionality of each referenced task.
- The object value associated with each task, if it has been provided during the task execution.

All the information is recovered from the reference task model XML file that actually contains the whole set of information about each task that has been modeled. Figure 40 illustrates an extract of the original (left side) XML scenario file, and the resultant (right side) XML scenario file after the process of preformatting.

<pre>... <step referencemodel="Inform a Flight Leg" role="subroutines" taskdate="Thu Apr 19 15:01:29 CEST 2018" taskdatelong="1524142889116"> <task taskid="t13"/> </step> <step referencemodel="Search Flights" role="tasks" taskdate="Thu Apr 19 15:01:40 CEST 2018" taskdatelong="1524142900016"> <task taskid="t23"> <stepObject objectID="6"/> </task> </step></pre>	<pre>... <step referencemodel="Inform a Flight Leg" role="subroutines" taskdate="Thu Apr 19 15:01:29 CEST 2018" taskdatelong="1524142889116"> <task taskid="t13" taskname="Set Departure Time Frame" optional="true" /> </step> <step referencemodel="Search Flights" role="tasks" taskdate="Thu Apr 19 15:01:40 CEST 2018" taskdatelong="1524142900016"> <task taskid="t23" taskname="Set Arrival Date" optional="false" /> <stepObject objectID="6" objectContent="Lun, Déc 10, 2018" /> </task> </step></pre>
---	--

Figure 40. Extract of an original (left side) and a resultant (right side) scenario XML files after the process of preformatting.

Besides preformatting the XML files of the extracted scenarios, our algorithm also adds, for each scenario, an equivalent scenario without the optional tasks. This is made due to a limitation in the current version of the HAMSTERS tool that does not allow to extract scenarios without the optional tasks. The tool necessarily includes both optional and non-optional tasks present in the model during the process of extracting scenarios. Thus, in order to obtain scenarios without the optional tasks, we algorithmically generate new scenarios eliminating all the tasks signalized

as optional in the set of scenarios extracted from HAMSTERS. Such new scenarios are named as “No Optional” followed by the original name of the scenario extracted from HAMSTERS. As a result, for each scenario extracted from HAMSTERS (necessarily including all optional tasks), we generate an additional similar scenario, but without all the optional tasks.

5.3.3.2. Automated Assessment

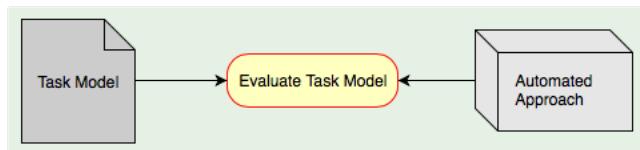


Figure 42. Activity of evaluating task models.

To illustrate how the assessing process is performed (Figure 42), we will follow the example already presented in the previous sections. As such, the left side of Figure 41 presents a scenario extracted from our HAMSTERS task model for modeling the User Story “Flight Tickets Search”. An extract of its before-preformatting XML source file is presented in the right side.

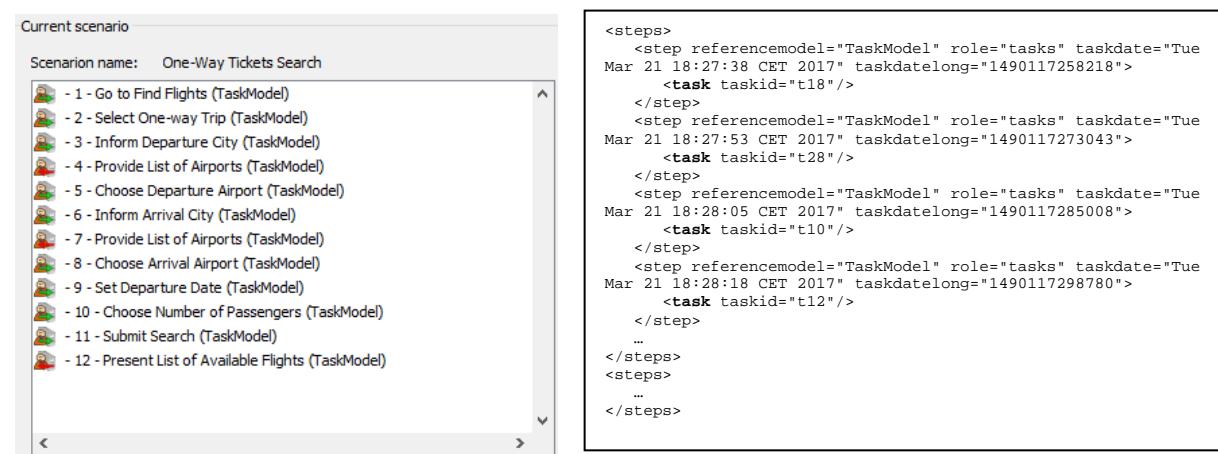


Figure 41. Example of scenario extracted from a task model and its XML source file.

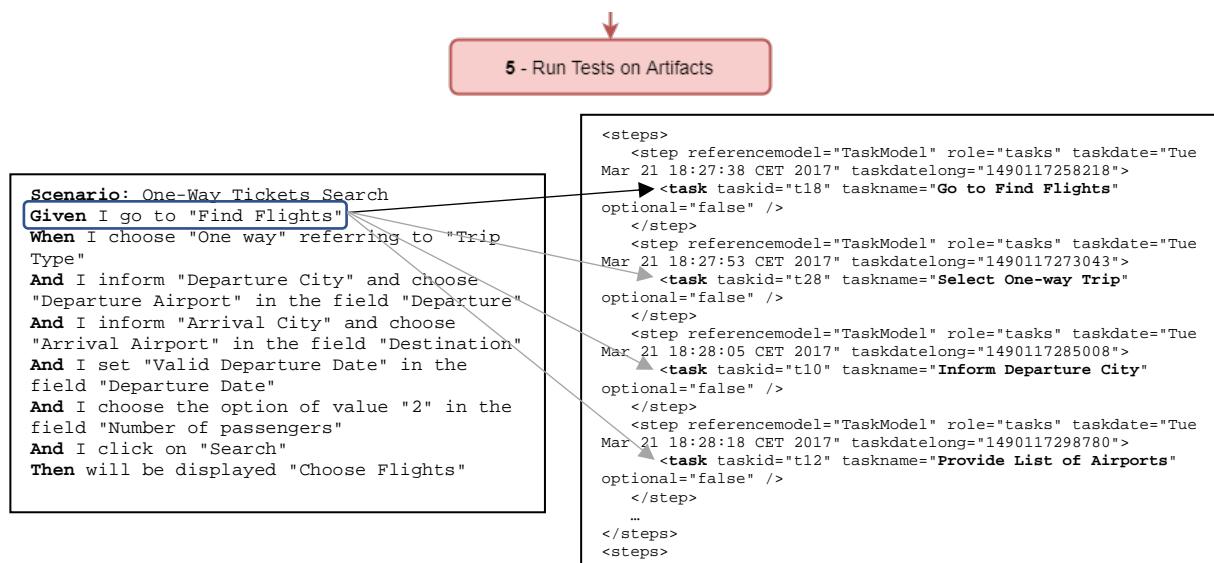


Figure 43. Checking consistency of tasks between US scenario and scenarios extracted from task models.

The process of consistency checking between US scenarios and scenarios extracted from task models consists of verifying, for each step in the US scenario, if there are one or more right correspondences for such a step in the XML source files of the scenarios extracted from the task models. To do so, as illustrated in Figure 43, our algorithm fixes a step in the US scenario (“*Given I go to Find Flights*” for example) and retrieves from the ontology the correspondent task to be verified in the task model, following the mapping presented in the section 5.3.2 (“*Go to Find Flights*” in the example). Then we parse each task of each scenario in the XML source file looking for one or more correspondences to the task retrieved from the ontology. If matches are found, then a list of matches is created, keeping the position in each scenario-task where the match has been found. The algorithm presented below in Figure 44 implements such a strategy.

```

foreach step from US Scenarios do
    taskToFind <- correspondent task from the ontology
    foreach task from each XML source file do
        if the attribute taskname is equal to taskToFind then
            ListOfMatches <- position(scenario,task)
        endif
    endforeach
endforeach
show ListOfMatches

```

Figure 44. Testing algorithm for assessing scenarios extracted from task models.

The results of testing are shown in a log indicating, for each step of the US scenario, if and where a given step has found an equivalent task in the XML file analyzed, and once it carries an object value associated, which value it is. In the example below, the first step (“*Given I go to Find Flights*”) of the scenario “One-Way Tickets Search” has found an equivalent task (i.e. a task named “Go to Find Flights”) in the first position of the first scenario (task 1). The second step, however, did not find a correspondent task once it was expected a task named “Choose Trip Type” and the task model brings a task named “Select One-way Trip” (task 2), so this represents an inconsistency in the model.

The third and fourth steps have a structure encompassing two user tasks, a first one to inform/select a departure city/airport, and a second one to inform/select an arrival city/airport from a list provided by the system. Both steps have not found correspondent tasks in the task model (respectively tasks 3/5 and 6/8), once it was expected respectively the tasks “Inform Departure”/“Choose Departure” when the task model actually brings “Inform Departure City”/“Choose Departure Airport”, and “Inform Destination”/“Choose Destination” when the task model actually brings “Inform Arrival City”/“Choose Arrival Airport”. The intermediate system tasks “Provide List of Airports” (tasks 4 and 7) in the scenario extracted from the task model have not been identified once there is not a correspondent step in the US scenario to represent them.

Tasks 9 and 10 are actually inverted in the US scenario. For the task 9, it was expected the task “Choose the option of value in the field Number of passengers” in the ninth position while it is actually found in the tenth position with the name “Choose Number of Passengers” (which would be an inconsistency anyway). For the task 10, “Set Departure Date” is expected in the tenth position when it is actually found in the ninth position, signalizing another inconsistency in the model. Finally, the task “Submit Search” has been correctly identified in the eleventh position, while the task “Present List of Available Flights” despite being correctly placed in the twelfth position, it was expected with the name “Display List of Available Flights” instead, which signalizes an inconsistency in the model. Table 15 summarizes such results.

```

Running story stories/search.storyConverted
Feature: Flight Tickets Search
(stories/search.storyConverted)
Narrative:
In order to obtain information about rates and times of the flights
As a user
I want to be able to search tickets, providing locations and dates.
Scenario: One-Way Tickets Search
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position:
1 >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position: 1 >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position:
1 >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position: 1 >>
Given I go to "Find Flights"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose Trip Type - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose Trip Type - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose Trip Type - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose Trip Type - Task not found! >>
When I choose "One way" referring to "Trip Type"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
When I inform "Departure City" and choose "Departure Airport" in the field "Departure"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
And I inform "Arrival City" and choose "Arrival Airport" in the field "Destination"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose Number of passengers - Task not
found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose Number of passengers - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose Number of passengers - Task not
found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose Number of passengers - Task not found! >>
When I choose the option of value "2" in the field "Number of passengers"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Set Departure Date - Found in Position:
9 - Associated Value: No Value >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Set Departure Date - Found in Position: 10 -
Associated Value: No Value >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Set Departure Date - Found in Position:
10 - Associated Value: No Value >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Set Departure Date - Found in Position: 9 -
Associated Value: No Value >>
And I set "Valid Departure Date" in the field "Departure Date"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Submit Search - Found in Position: 11 >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Submit Search - Found in Position: 11 >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Submit Search - Found in Position: 11 >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Submit Search - Found in Position: 11 >>
When I submit "Search"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Display List of Available Flights - Task
not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Display List of Available Flights - Task not found!
>>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Display List of Available Flights - Task
not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Display List of Available Flights - Task not found!
>>
Then will be displayed "List of Available Flights"

```

As scenarios in User Stories and scenarios in task models may be ordered differently, the algorithm checks the whole set of XML files to ensure we are looking for all the instances of the searched task. So, notice that the log of results presented above shows, for each step of US scenario, the results of searching in each XML scenario file (“.scen”). Each line of results brings then:

- the name of the scenario in which the search has been carried out,
- the task name that has been searched for,
- the position in which the task has been found (if so), otherwise is shown the message “Task not found!”, and

- the object value associated with each task (if any), otherwise is shown the message “No Value”.

Due to that, if there are several XML files of scenarios, the results in the log will show where a correspondent task has been found in each one of them. A consequence of such a strategy is that the process of analyzing if a given task is correctly positioned in the evaluated scenarios is made manually after getting the whole log of results.

Step in US scenario	Expected	Task in the XML source file	Test Result
Given I go to “Find Flights”	Go to Find Flights	1 - Go to Find Flights	V
When I choose “One way” referring to “Trip Type”	Choose referring to Trip Type	2 - Select One-way Trip	X
And I inform “Departure City” and choose “Departure Airport” in the field “Departure”	Inform Departure	3 - Inform Departure City	X
	-	4 - Provide List of Airports	X
	Choose Departure	5 - Choose Departure Airport	X
And I inform “Arrival City” and choose “Arrival Airport” in the field “Destination”	Inform Destination	6 - Inform Arrival City	X
	-	7 - Provide List of Airports	X
	Choose Destination	8 - Choose Arrival Airport	X
And I choose the option of value “2” in the field “Number of passengers”	Choose the option of value in the field Number of passengers	9 - Set Departure Date	X
And I set “Valid Departure Date” in the field “Departure Date”	Set Departure Date	10 - Choose Number of Passengers	X
And I submit “Search”	Submit Search	11 - Submit Search	V
Then will be displayed “List of Available Flights”	Display List of Available Flights	12 - Present List of Available Flights	X

Table 15. Checking consistency of tasks between US scenario and scenarios extracted from task models.

5.3.3.3. Tool Support

The algorithm we have just described for testing task models (Figure 44) has been implemented in the Eclipse IDE for Java EE. The project has been structured in two packages. The first one encompasses the classes for implementing the solution. As shown in Figure 45, this package contains four classes: MySteps, MyTest, MyXML and PrepareFiles. MySteps implements the mapping between the Common Steps described in the ontology and the assertion that should be made when checking scenarios from task models. MyXML implements methods for parsing scenario files extracted from task models in their XML files. MyTest is the JUnit class that triggers the set of User Stories that have been selected for testing. Finally, PrepareFiles is the class in charge of preformatting the scenario source files extracted from task models, as described in section 5.3.3.1.

The second package encompasses the resources demanded for running the tests. In the folder “stories”, we have the whole set of User Stories text files that have been specified for the project. Even being text files, each User Story file must be named with a “.story” extension. In the example, the project has one single User Story, with different scenarios for testing a given feature. The folder “scenarios” contains the current scenario’s XML files extracted from task models under testing, before and after the process of preformatting described in section 5.3.3.1. Finally, the folder “task models” keeps the reference XML source files for the task models under testing. Such files are useful to allow the process of preformatting.

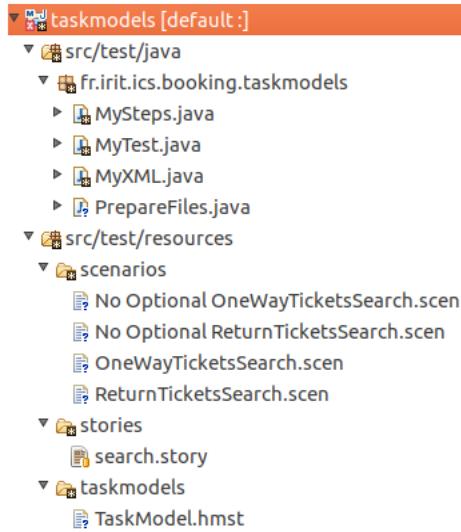


Figure 45. File tree for the implementation of task model assessment.

Figure 46 represents the flow of calls we have designed in our algorithm for running a battery of tests on task model scenarios. The flow starts with the class “MyTest.java”. First of all, this class instantiates an object from “PrepareFiles.java” (flow 1) in order to trigger the process of preformatting mentioned before. Such a process runs on the package of task model scenarios (flow 2), naming the extracted tasks and adding useful complementary information for testing. For that, the process asks the reference source file (.hmst) of the correspondent task model mentioned by each task in the scenario. After getting the scenario files formatted, “MyTest.java” includes the User Story (or the set of User Stories) that will be tested (flow 3).

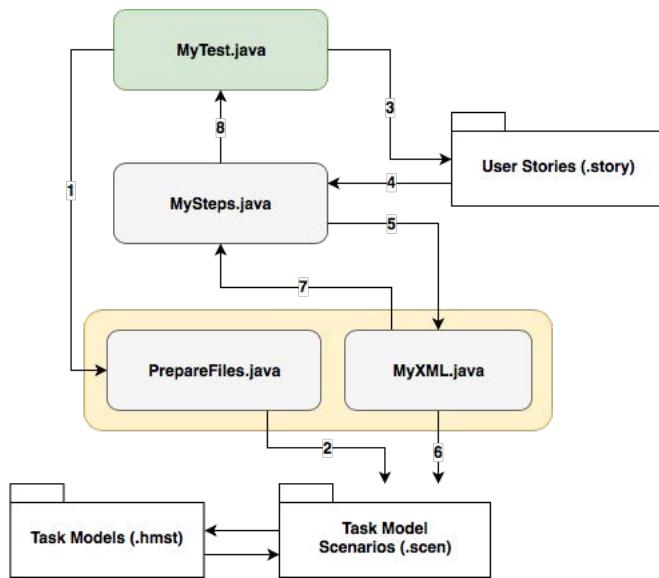


Figure 46. Flow of calls for running tests on task model scenarios.

Each one of the steps in the User Story under testing makes a call to the class “MySteps.java” (flow 4) that knows which behaviors are supported by the ontology. Based on the behavior referenced by the step, this class makes a call to the class “MyXML.java” (flow 5) in charge of parsing all the set of task model scenarios (flow 6). This parsing aims to check if the behavior addressed by the step is also present in at least one of the scenarios extracted

from the task models. The result of this parsing is then returned to the class “MySteps.java” (flow 7). At this point, based on the algorithm presented in the previous section, a list of all the matches found during the parsing for each step is presented as a result. Finally, the class “MySteps.java” returns the result to the class “MyTest.java” (flow 8) that made the original call.

Notice the independence of the components assigned at the core of the structure represented in Figure 46 (highlighted in yellow). Those components are related to the particularities of test implementation for HAMSTERS task models and scenarios. As mentioned before, “PrepareFiles.java” is in charge of preformatting the extracted scenario files and reading the reference source file of task models, while “MyXML.java” is in charge of parsing the scenario files, searching for the elements under testing. Therefore, we deliver a flexible architecture allowing, in the future, that task models and scenarios modeled by other modeling tools (or even by other versions of HAMSTERS) could also be tested by just implementing new interfaces for this core.

5.3.3.4. Setup and Running

Considering the presented architecture, to setup and run a battery of tests, we must:

- Place the set of task model scenario files (“.scen”) that will be tested in the package “Task Model Scenarios”.
- Place the set of task model files (“.hmst”) that will support the test in the package “Task Models”.
- Place the set of User Stories files (“.story”) that will be tested in the package “User Stories”.
- Indicate in the “MyTest” class which User Story will be tested, or which folder (“/stories”) contains all the User Stories that will be tested.
- Run the “MyTest” class as a JUnit Test.

```
@Test  
public void testAllStories() throws Throwable {  
    eng.addSteps(new MySteps());  
    eng.addStories("/stories/search.story");  
    eng.run();  
}
```

Figure 47. “MyTest” class indicating the file “search.story” for running.

Thus, for running the tests, the MyTest class is triggered. This JUnit class specifies exactly which User Story (or which set of User Stories) will be run. Figure 47 illustrates the implementation for running the User Story “Flight Tickets Search” (in the file “search.story”). This story has the following scenarios:

<pre>User Story: Flight Tickets Search Narrative: In order to obtain information about rates and times of the flights As a user I want to be able to search tickets, providing locations and dates. Scenario: One-Way Tickets Search Given I go to "Find Flights" When I choose "One way" referring to "Trip Type" And I inform "Departure City" and choose "Departure Airport" in the field "Departure" And I inform "Arrival City" and choose "Arrival Airport" in the field "Destination" And I choose the option of value "2" in the field "Number of passengers"</pre>
--

```

And I set "Valid Departure Date" in the field "Departure Date"
And I submit "Search"
Then will be displayed "List of Available Flights"

Scenario: Return Tickets Search

Given I go to "Find Flights"
When I choose "Round trip" referring to "Trip Type"
And I inform "Departure City" and choose "Departure Airport" in the field "Departure"
And I inform "Arrival City" and choose "Arrival Airport" in the field "Destination"
And I choose the option of value "1" in the field "Number of passengers"
And I set "Valid Departure Date" in the field "Departure Date"
And I set "Valid Arrival Date" in the field "Arrival Date"
And I submit "Search"
Then will be displayed "List of Available Flights"

```

Finally, Figure 48 shows the console with the results of tests running the two scenarios specified in this story above. Notice that, as described in the previous section, for each step of the US scenario, it has been shown where some correspondent task has been found and which value was associated to it (if any).

```

Markers Properties Servers Data Source Explorer Snippets Problems Console Progress Synchronize Call Hierarchy

<terminated> MyTest [8] [JUnit] /usr/local/java/jdk1.8.0_162/bin/java (May 14, 2018, 5:54:25 PM)
Running story stories/search.storyConverted
Feature: Flight Tickets Search
(stories/search.storyConverted)
Narrative:
In order to obtain information about rates and times of the flights
As a user
I want to be able to search tickets, providing locations and dates.
Scenario: One-Way Ticket Search
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position: 1 >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position: 1 >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position: 1 >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Go to Find Flights - Found in Position: 1 >>
Given I go to "Find Flights"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose referring to Trip Type - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose referring to Trip Type - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose referring to Trip Type - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose referring to Trip Type - Task not found! >>
When I choose "One way" referring to "Trip Type"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose Departure - Task not found! >>
When I inform "Departure City" and choose "Departure Airport" in the field "Departure"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose Destination - Task not found! >>
And I inform "Arrival City" and choose "Arrival Airport" in the field "Destination"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Choose the option of value in the field Number of passengers - Task not found! >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Choose the option of value in the field Number of passengers - Task not found! >>
<< Scenario: No Optional ReturnTicketsSearch.scen - Searched Task: Choose the option of value in the field Number of passengers - Task not found! >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Choose the option of value in the field Number of passengers - Task not found! >>
When I choose the option of value "2" in the field "Number of passengers"
<< Scenario: No Optional OneWayTicketsSearch.scen - Searched Task: Set Departure Date - Found in Position: 9 - Associated Value: No Value >>
<< Scenario: ReturnTicketsSearch.scen - Searched Task: Set Departure Date - Found in Position: 10 - Associated Value: No Value >>
<< Scenario: OneWayTicketsSearch.scen - Searched Task: Set Departure Date - Found in Position: 11 - Associated Value: No Value >>

```

Figure 48. Console after running the User Story “Flight Tickets Search”.

5.3.4. Towards an Alternative to the Extraction of Scenarios

Task models can be tricky to manipulate once the number of possible scenarios can scale exponentially due to the complexity of the model. Different operators, the presence of optional tasks, the number of times an iterative task can be executed, etc. make the extraction of scenarios for testing a very complex activity. Campos et al. (Campos *et al.*, 2017) illustrate this problem and propose a catalog of strategies for modifying the models in order to manage the complexity of the resultant set of extracted scenarios. An easy-to-see consequence of such kind of strategy is that models are not fully manipulated, i.e. the reference model for extracting test scenarios is a simplified instance (a subset) of the original model. Consequently, several nuances of modeling (such as the use of multiple operators, non-interactive tasks, etc.), which allow task models being a rich representation of human activities when interacting with the system, are lost and cannot be verified or even taken into account when obtaining scenarios.

In order to exemplify this problem, Figure 49 retakes an example of our current approach for extracting scenarios from task models. The model presents a short extract of some tasks (1 in Figure 49) involved in the process of booking flight tickets through a generic flight booking system. Therein, an abstract task named “Provide Data” generalizes a sequence of 5 tasks that can be performed in any other. This attribute is signalized by the operator “Order Independent” ($|=|$) placed between “Provide Data” and the other 5 tasks. Thus, one of the possible scenarios that could be extracted from this model is presented further (3 in Figure 49). Therein, tasks are performed in the order they are visually presented in the model, i.e. first the user informs a destination and a departure, then he/she chooses the number of passengers, sets the departure date, and finally chooses his/her trip type.

Notice that the XML source file of the extracted scenario (4 in Figure 49) is just a sequential description of tasks in the model that have been settled for execution. The file brings for each task only a reference for its ID (it does not even bring the name of the task), the source task model, and the date/time of execution. The XML source file of the task model itself (2 in Figure 49) is, on the other hand, a richer description of task modeling elements, including tasks of several types, operators, constraints related to the number of iterations each task supports, tasks that are optional, maximum and minimum time of execution, levels of criticality and so on.

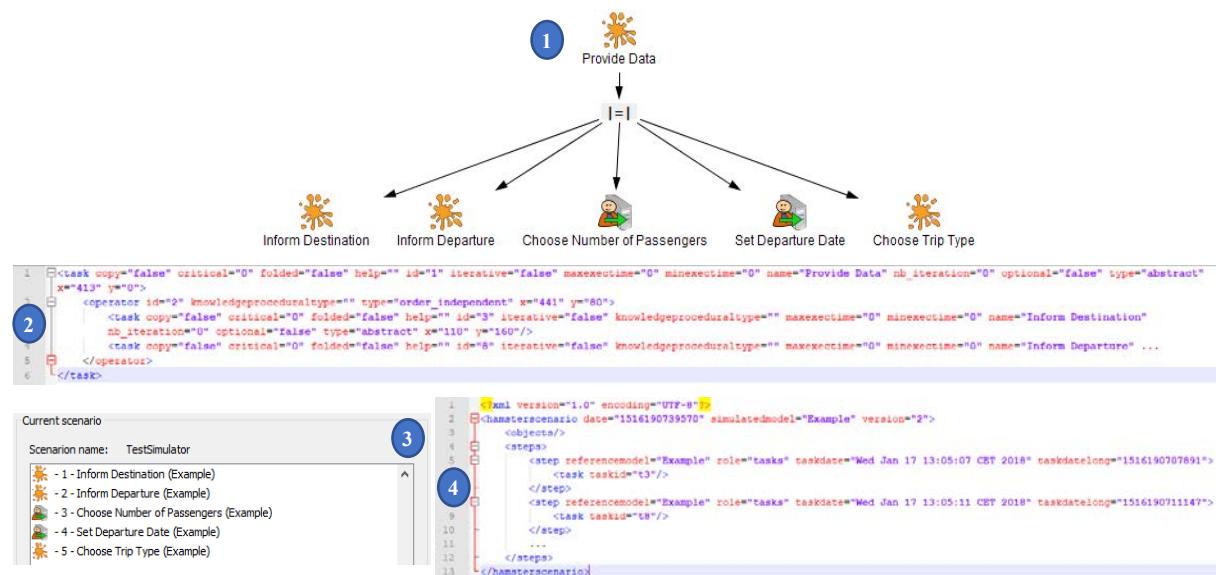


Figure 49. Task model (1), extracted scenario (3), and their respective source files (2 and 4).

Therefore, we can easily realize that the manipulation of XML source files of task models brings us a full range of challenges. For example, as pointed by Campos et al. (Campos et al., 2017), the presence of “order independent” operators between subtasks is a major contributor to the state explosion in the state machine generated from a task model. Considering a simple example in Figure 49, although the task model has only five subtasks following the abstract task “Provide Data”, the resultant number of possible combinations of tasks (resulting in scenarios) is equal to 120. This happens because we must consider all the permutations of the five tasks’ execution. By following all the leaves in a task model with multiple operators, we notice that the number of possible scenarios for extraction gets exponential in function of the types of these operators. That is the reason by which authors working with task model exploitation for generating scenarios or test cases usually control such an extraction, in order to reduce the

complexity and the resultant number of combinations. This becomes especially challenging if task models specify collaborative activities with several instances of the same role.

We have also followed a strategy based on the extraction of scenarios from task models for obtaining test scenarios to check the quality of models, but unlike other approaches, we have not controlled such an extraction, which allowed us to keep important aspects of interaction. However, as current tools do not allow us to automatically extract, from a given model, the full set of possible scenarios for execution, this process is made manually by following all the achievable paths and formatting them to get the resultant scenario prepared for testing. Figure 50 illustrates the flow of activities we have performed so far to obtain scenarios for testing based on the current approaches in the literature. Notice that the source file of a task model is manipulated only for extracting scenarios (continuous black line at the top). Such a feature is usually included in tool-supported notations for designing task models. After such an extraction, the resultant source files of scenarios are manipulated and formatted to obtain a given scenario for testing (continuous black line at the right).

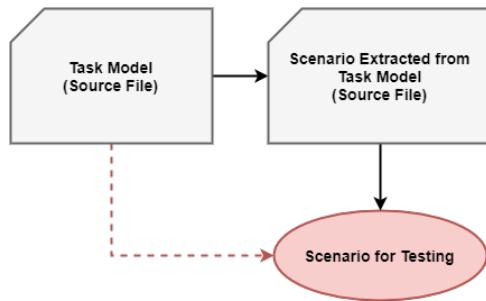


Figure 50. Flow of activities to get scenarios for testing.

This approach, regardless keeping important aspects of the interaction during the extraction process, has limitations once it is still fully dependent on the extraction of scenarios, i.e. it does not allow us to manipulate the source file of task models directly. An approach that does not necessarily pass through extracted scenarios (dotted red line at the left) would allow us to manipulate the model with its full capabilities, opening a wide range of opportunities to assess the quality of such models and to obtain not-simplified scenarios for testing. In short, being able to manipulate and check the consistency of task models directly in their source files, instead of passing by the process of extracting scenarios, could represent a crucial step forward a solution that includes a complete and not-simplified assessment strategy for artifacts modeling user requirements.

5.4. Conclusion

This chapter presented our approach to assess task models by following a strategy in which task models are designed, scenarios are extracted from them, and then User Stories are written and formatted based on such extracted scenarios. This is one of the strategies we designed to perform our approach. As presented in chapter 3, alternatively, we can write formatted User Stories before designing our task models. It means that depending on the characteristics of the project, either the User Stories can support the design of task models, or the task models (by means of their extracted scenarios) can support the writing of User Stories.

Despite the limitations related to the process of extracting scenarios from task models, the strategy for assessment we present in this chapter has many advantages over co-execution

approaches. When opting for a static analysis of the source files, we gain in performance and availability of tests. Specially in environments requiring a high-availability of tests to be executed continuously along multiple iterations, static approaches benefit from an instantaneous consistency checking analyzing several hundreds of scenario files at the same time. Co-execution approaches have the benefit of allowing running models simultaneously with a visual feedback at real-time about the correspondence of entities that are being assessed in each model. However, such approaches demand a high investment to prepare the models before, annotating the source code/files or even modifying its structure to support the co-execution. Besides that, as the great benefit of co-execution is providing a visual feedback during the execution signalizing which entity is being assessed in each model at a given time, this process is usually slow and require an evaluation being conducted manually to reveal its benefits.

Another benefit of our approach when compared with co-execution ones is that we defined an open and flexible architecture where different notations and tools for designing task models could fit in the future. For that, it is enough to implement a new core interface for describing the way such notations and tools deal with tasks and scenarios, and how they can be identified in their source files.

Finally, strategies for running automated tests over software artifacts indeed define a step forward within the process of software verification. Such a process, that is usually conducted manually by just inspecting or reviewing the artifacts in an attempt to identify inconsistencies and modeling errors, can benefit from an automated approach giving high-available instantaneous feedback about the consistency of artifacts with the user requirements all along the iterations.

The chapter 8 employs our automated approach in a large case study including the design of task models to the booking system of business trips in our institute. The chapter details a broad set of inconsistencies our approach is able to identify and provides results about its potential. The next chapter follows presenting our approach for multi-artifact testing detailing our strategy for assessing user interface prototypes in different levels of refinement. As each artifact has its own characteristics, the strategy slightly differs from the one for task models we have just presented in this chapter. However, as an integrated approach, the same User Stories will be assigned to assess the set of UI prototypes in different levels of abstraction in order to keep a consistent model-checking approach for interactive systems.

5.5. Resultant Publications

Silva, T. R. & Winckler, M. (2017). A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts. In: Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems (IHC), pp. 21-30. ACM. DOI: <http://doi.org/10.1145/3160504.3160506>. (Silva and Winckler, 2017)

Silva, T. R., Hak, J. L. & Winckler, M. (2016). An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development. Complex Systems Informatics and Modeling Quarterly, 1 (7), pp. 81-107. DOI: <http://doi.org/10.7250/csimg.2016-7.05>. (Silva, Hak and Winckler, 2016a)

Chapter 6

Modeling and Assessing User Interfaces: From Prototypes to Final UIs

Summary

This chapter details our strategy for modeling and assessing user interface prototypes following our approach presented in chapter 3. The chapter begins by resuming the case study proposed previously, assuming that Balsamiq prototypes will be used to design the user interface in a first stage of refinement. By following this, we present firstly how to produce UI prototypes already consistent with the set of user requirements specified previously. The example of UI prototype we explore is based on the searching flight activity already modeled in the task model presented in the previous chapter.

In the second section, we present how our previous developed ontology can support the development of prototyping tools able to produce consistent UI artifacts. PANDA is a tool supporting such a mechanism. It provides a full pallet of widgets based on the presentation layer of prototypes described in the ontology, and a full range of behavior properties, based on the common interactive behaviors, also described in the ontology. PANDA prototypes also feature a state machine for modeling the dialog, exactly as described in the ontology.

The third section describes how we perform tests on fully implemented user interfaces by using an integrated multiplatform framework. This framework allows designing automated acceptance testing with low implementation efforts. The fourth section discuss how our approach supports the assessment of evolutionary UI prototypes (using PANDA and/or other tools), and how it could keep them consistent along the software development. Finally, the fifth and last section concludes the chapter pointing out advantages and limitations of this approach.

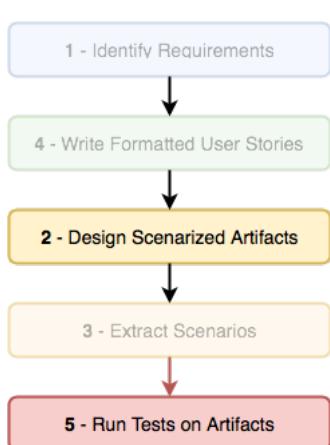


Figure 51. Another alternative for performing our approach.

In iterative processes, the design of user interfaces can evolve all along the software development process as a result of requirements evolution and change, or the need of understanding and validating a given interpretation of requirements (Wood and Kang, 1992). While the beginning of the project usually requires a low-level of formality with UI prototypes being hand sketched to explore design solutions and clarify user requirements, the development phase requires more refined versions frequently describing presentation and dialog aspects of interaction. Full-fledged versions of user interfaces are generally produced only later in the design process, and frequently corresponds to how the user “see” the system. In the users’ point of view, if some feature is not available on the user interface, this feature does not exist for them. Besides that, acceptance testing is generally conducted by users on full-fledged versions of user interfaces, which should be fully functional at this stage.

In this chapter, we adopt a use of user interface prototypes serving as an early and evolutive design artifact for modeling aspects of functional user requirements. The

evolution process of such artifacts brings the need of assessing them in their multiple stages of development. Our strategy concerning the assessment of user interface prototypes consists, like for task models, in checking their consistency with respect to a previously-defined requirements specification. Unlike the chapter 5 however, in this chapter, we present the approach by following a different alternative (Figure 51). As formatted User Stories have already been designed from the scenarios extracted from task models, these same stories will be used to design UI prototypes in an attempt to get such prototypes already consistent with the user requirements. The process of extracting scenarios from task models has defined which scenarios would be considered for testing, therefore as such scenarios have already been defined and gave rise to the formatted User Stories that will be used, this activity will be skipped for the UI prototypes. Thus, in this chapter, we will in the sequence only: design the user interface prototypes (as part of the scenarized artifacts) (2), and finally run tests on the artifacts (5).

Like task models, user interface prototypes can be designed through a diverse set of notations and tools. For being assessed under our approach though, they only need to comply with the premise of exporting the source files of prototypes in a markup language. As any other scenarized artifacts, user interface prototypes could perfectly be a candidate to have scenarios extracted from them. If it is such a case, such scenarios should also be provided with their source files in a markup language. In this case, they may also serve as input to get formatted User Stories following the strategy presented for task models in chapter 5.

Like task models again, although in theory user interface prototypes in any notation may be assessed since they comply with the premise stated above, our implementation should be adapted to understand the formalism used by such notations to describe the UI prototype and eventually the scenarios extracted from them. Our strategy for testing performs a static assessment of the source files of prototypes by means of a syntactic and semantic analyzes of such files. As signalized in chapter 5, an advantage of this approach is that, unlike co-execution approaches where both artifacts under testing should be prepared for assessment by annotating or modifying their source files, with our approach we have no need to intervene in the source files of the target artifacts, i.e. artifacts do not need to be prepared for testing by designers, so both user interface prototypes and requirements specifications can be assessed in their original state.

For the demonstration we propose in this chapter, we make use of user interface prototypes designed by Balsamiq once the notation and tool fit our premise stated above. Balsamiq exports its prototypes using the XML standard, a well-adopted markup language, so recognized by our approach. The task modeling and the extraction of scenarios that will be presented hereafter has been made by using the Balsamiq tool, whilst the implementation of the assessment has been made by using the respective XML source files produced by the Balsamiq tool for each model.

In a last stage of refinement, we also make of use of final user interfaces to assess user requirements with respect to the definitive aspect of the interaction. We call final user interfaces (final UIs), the fully functional versions of a UI prototype implemented in a given programming language for a given platform. Unlike task models and Balsamiq prototypes, the assessment of a final UI is made by dynamically running tests on its presentation layer with the aid of external testing frameworks. Our premise for assessing such final UIs is then the availability of an external testing framework able to run tests on a given environment. So far, our approach can implement integration with Selenium WebDriver³ for assessing user interfaces of web applications, which will

³ <https://www.seleniumhq.org/projects/webdriver/>

be used in this chapter). For assessing final UIs implemented for other environments such as desktop or mobile applications, our approach should integrate with other testing frameworks.

We have also experienced the use of our ontology to develop a new prototyping tool in order to allow the design of prototypes already consistent with the set of interactive behaviors defined in the ontology. PANDA tool allows the description of prototypes with a more refined description of the interaction when compared with Balsamiq. This tool is presented in the section 6.2. We also describe in the next sections our strategy for performing tests since the low-refined Balsamiq wireframes until full-fledged versions of the user interface running on the web.

6.1. Starting with Balsamiq Wireframes

For designing UI prototypes in a low level of refinement, we have chosen the sketches produced by Balsamiq Mockups⁴. Balsamiq is a rapid wireframing tool that reproduces the experience of sketching on a whiteboard but using a computer. Balsamiq has a large set of handmade-style UI elements for composing a user interface sketch. Figure 52 gives an overview of such elements.

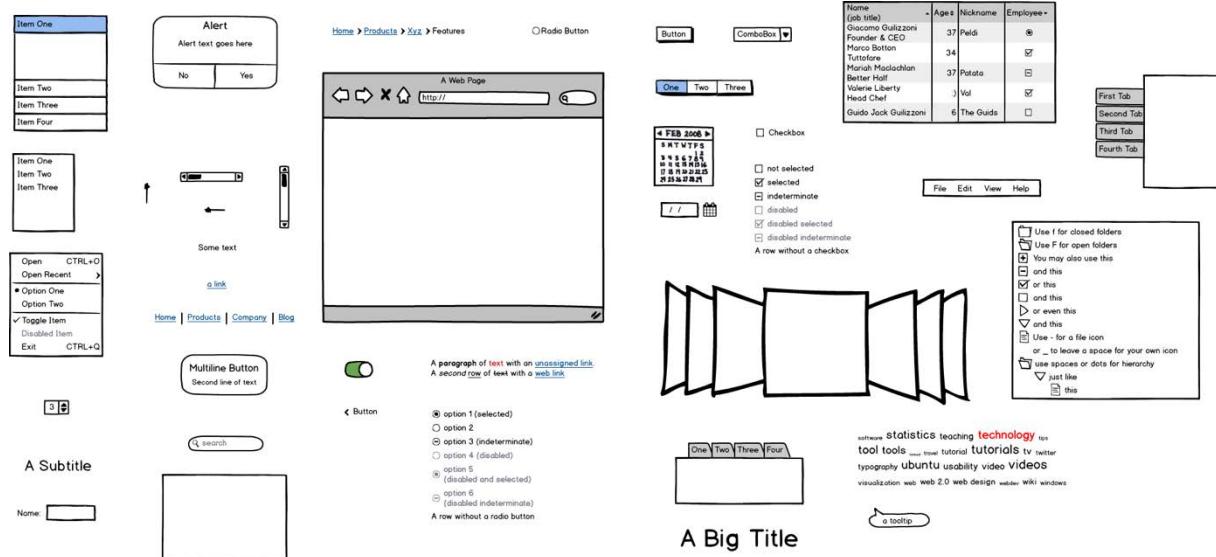


Figure 52. Balsamiq handmade-style UI elements.

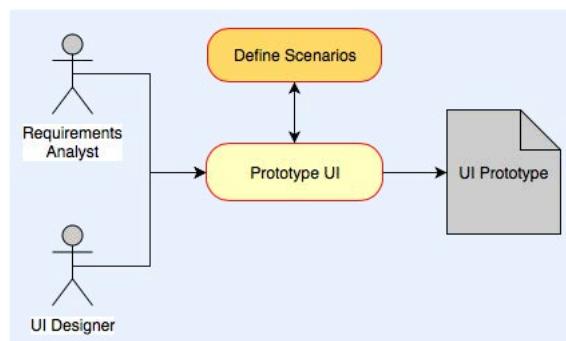


Figure 53. Activity of prototyping UIs.

By following the steps of the scenario “One-Way Tickets Search” and consulting the ontology to identify matching interactive elements, the prototype can be designed already considering the set of interactive elements supported by each

Figure 54 presents the scenario “One-Way Tickets Search” (formatted in chapter 5 after extracting scenarios from the task models), supporting the development of a Balsamiq sketch prototyped for the User Story “Flight Tickets Search”. The activity of design the prototypes is performed in collaboration by Requirements Analysts and UI Designers (Figure 53).

By following the steps of the scenario “One-Way Tickets Search” and consulting the ontology to identify matching interactive elements, the prototype can be designed already considering the set of interactive elements supported by each

⁴ <https://balsamiq.com/>

behavior. For example, when consulting the ontology, we find that the behavior “goTo” in the first step (“I go to ‘Find Flights’”) is supported only by the interaction element Browser Window. Thus, the designer has no other option to address this behavior. Indeed, in the prototype, it has been used a Browser Window for this behavior. On the other hand, the fifth step (“I set ‘Valid Departure Date’ in the field ‘Departure Date’”) addresses the interaction element “Departure Date” that refers in the prototype to the Calendar used for picking up a date of departure. The behavior “setInTheField” is also supported by Dropdown Lists, Text Fields and Autocompletes. Thus, the designer could have picked any of them instead, but not a Button, for instance, once it does not support the behavior “setInTheField”.

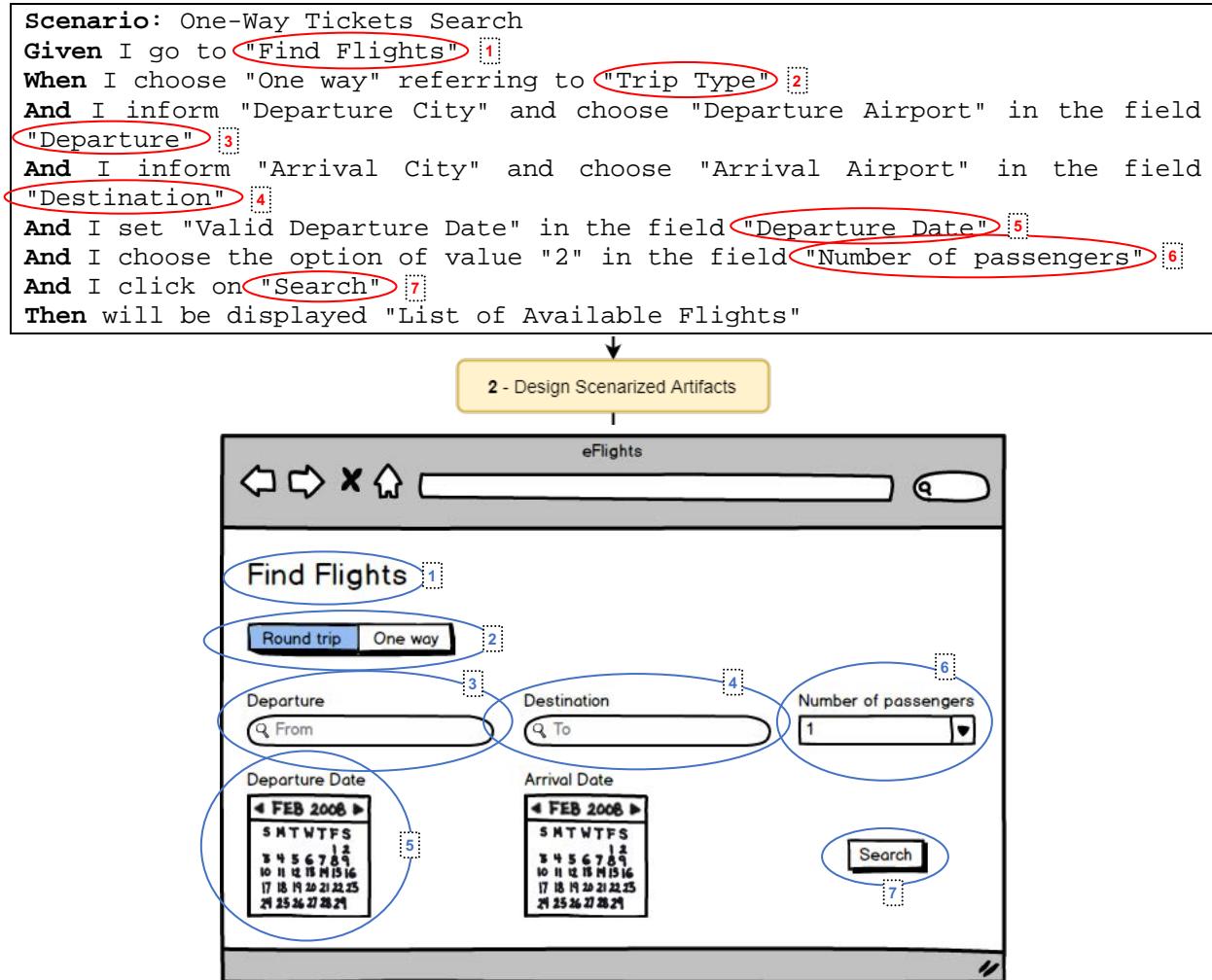


Figure 54. Sketch for the User Story “Flight Tickets Search” built from the scenario “One-Way Tickets Search”.

The second step addresses the interaction element “Trip Type” that refers to the Link bar used for choosing between a one-way and a round trip. The third and fourth steps addresses the interaction elements “Departure” and “Destination” that refers to the Text Fields, but with a searching feature. It means that this element supports an operation auto-complete where, with a single interaction, the user attains to inform some partial text and (based on the instant matching results) choose the desired option. The sixth step addresses the interaction element “Number of passengers” that refers to the Combo Box used for choosing the number of passengers in a finite list. Finally, the seventh step addresses the interactive element “Search” that refers to the Button used for submitting the search.

6.1.1. Test Implementation

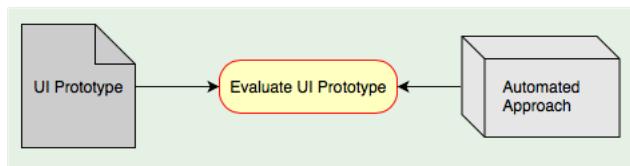


Figure 55. Activity of evaluating UI prototypes.

Just like task models, there are multiple notations and tools with different implementations for designing and modeling UI prototypes (Silva *et al.*, 2017). Among these multiple tools, we have chosen to implement a proof of concept with Balsamiq in its current version (2.2.28) once it fits our premise for getting user interface prototypes ready for testing. However, as we have done for the implementation of task models, we have designed a flexible and open architecture where other notations and tools could benefit from our approach by just implementing a new java class in accordance with their own patterns to implement and model prototypes.

The assessing of UI prototypes is an automated process as illustrated in Figure 55. The source code of Balsamiq prototypes is provided by the use of an XML specification. Thus, our strategy for testing such prototypes is parsing their XML source files, looking for UI elements that match the ontology description for each mapped behavior. Then the first step for assessing such prototypes is to get from the ontology the list of UI elements that support the behavior under testing. Taking the step “*And I set ‘Valid Departure Date’ in the field ‘Departure Date’*” as an example, by parsing the ontology OWL file, we find that the associated interactive behavior “#setInTheField” is supported by the UI elements “Dropdown List”, “Text Field”, “Autocomplete” and “Calendar”, when performing an “Action” (Then) or an “Event” (When) in a state machine transition.

After getting such a list of supported UI elements, we pursue to analyze the Balsamiq XML file to identify firstly if a field named “Departure Date” exists. This is made by reading the tag “<text>” identified in the parent tag “<controlProperties>” for a given “<control>” element. If such a field exists, i.e. there is a tag “<text>” carrying its name (case insensitive), so we retrieve which interaction element is associated with it. At this point, we have implemented a reference file containing the mapping between the abstracted interaction elements in the ontology and the Balsamiq concrete implementation of such elements.

In our sketch, we can notice that the field “Departure Date” has been modeled with a “Calendar” (extract in Figure 57), i.e. the UI designer has chosen the UI element “Calendar” to attend the field “Departure Date”. Thus, by checking the list of supported UI elements, we find that the behavior “#setInTheField”, addressed by the field “Departure Date”, is supported by a “Calendar” element, so the test would pass. If other elements than “Dropdown List”, “Text Field”, or “Autocomplete” had been chosen, the test would fail.

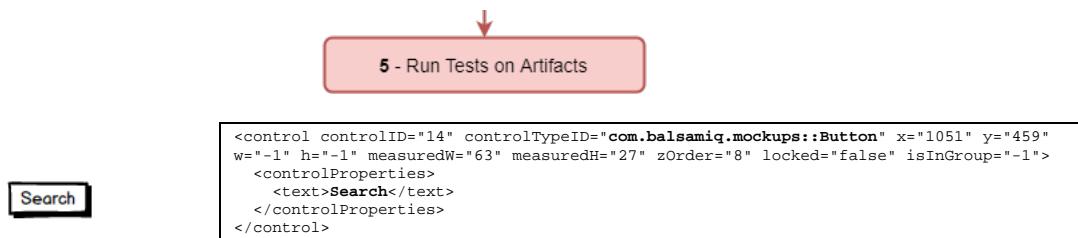
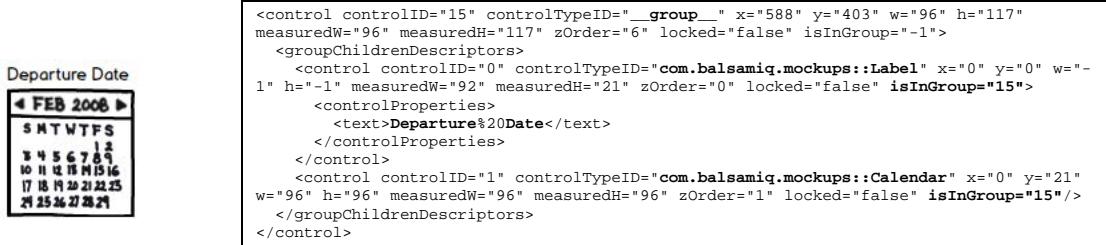


Figure 56. Button “Search” and its XML source file.



```

<control controlID="15" controlTypeID="__group__" x="588" y="403" w="96" h="117"
measuredW="96" measuredH="117" zOrder="6" locked="false" isInGroup="-1">
  <groupChildrenDescriptors>
    <control controlID="0" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="-
1" h="-1" measuredW="92" measuredH="21" zOrder="0" locked="false" isInGroup="15">
      <controlProperties>
        <text>Departure%20Date</text>
      </controlProperties>
    </control>
    <control controlID="1" controlTypeID="com.balsamiq.mockups::Calendar" x="0" y="21" w="96" h="96" measuredW="96" measuredH="96" zOrder="1" locked="false" isInGroup="15"/>
  </groupChildrenDescriptors>
</control>

```

Figure 57. Grouped field “Departure Date” and its XLM source file.

Balsamiq has two methods for representing UI elements on its XML source files. They can be directly assigned with a unique controlID (Figure 56) or be part of a group that encompasses a label and the UI element itself (Figure 57). In the first case, the label “Search” is directly associated with the element “Button” itself (com.balsamiq.mockups::Button). In the second case, we can notice the label for “Departure Date” is part of a group (isInGroup=’15’). In the same group, but with other controlID, we find the element “Calendar” itself (com.balsamiq.mockups::Calendar). Our testing algorithm implements then a solution that covers both situations. The algorithm presented in Figure 58 illustrates this implementation.

When looking for matching elements, the algorithm identifies which Balsamiq method has been used to design the element. If the parent tag is a label, it means that the element is part of a group that contains the element itself in a sibling tag. This sibling tag is then identified by reading the attribute “isInGroup”. If the parent tag is not a label, so it is already the element itself. After identifying it, the algorithm checks if some of the UI elements received from the ontology matches with the element from the prototype that is being investigated. If so, the variable “numTasks” is increased by one. After investigating the whole set of tags, the value of this variable is returned and must be equal to “1”, which means only one UI element for representing the “fieldname” has been found. If this value is equal to “0”, it means that no UI element has been found in the prototype with that “field-name”, while if it is greater than “1”, it means that more than one UI element has been found with the same “fieldname”. In both cases, the algorithm identifies the failure and the test does not pass. This process is conducted for each step of the scenario.

```

foreach step from US Scenarios do
  supportedUIElements <- correspondent UI Elements from the ontology
  fieldName <- name of the UI Element from the step
  foreach UI Element from the Balsamiq prototype do
    if the attribute text is equal to fieldName && is not in group then
      if the attribute controlTypeID is equal to one of the
        supportedUIElements then
          numElements++
    endif
    else if the attribute text is equal to fieldName && is in group then
      if the attribute controlTypeID of some member of the group is
        equal to one of the supportedUIElements then
          numElements++
    endif
  endforeach
  endforeach

  if numElements == 1 show Success
  else show Fail

```

Figure 58. Testing algorithm for assessing UI prototypes.

Notice that for prototypes at this level of refinement, we only assess the presentation component. We are not considering for testing at this level the dialog modeling and the consequent dynamic aspect of the interaction. It means that to check the consistency of the UI elements modeled in the prototype, we only consider the presence (or the absence) of the right interaction elements on the screen where the interaction is supposed to occur. Behaviors that perform a state transition (e.g. navigating from one screen to another or getting mock values from the fields as a result of an interaction) are not being taken into account in the results. The next section presents our strategy for considering the dialog aspect of prototypes that are one step forward in the level of refinement.

6.1.1.1. Running Tests

Figure 60 represents the flow of calls we have designed in our algorithm for running a battery of tests on Balsamiq prototypes. The flow starts with the class “MyTest.java” that is a JUnit class in charge of triggering the battery of tests (its content is illustrated in Figure 59). This class indicates which files will be used for testing (flow 1). There files are distributed in two packages. The first one contains the User Story files (where are the scenarios for testing), and the second one contains the Balsamiq UI Prototypes files (that are the BMML source files of Balsamiq prototypes). So, in the example below, it has been indicated for testing the story “Flight Ticket Search.story” on the Balsamiq UI prototype “Book Flights.bmml”.

```
@Test  
public void testAllStories() throws Throwable {  
    eng.addSteps(new MySteps("src/test/resources/lfprototypes/Book Flights.bmml"));  
    eng.addStories("/stories/Flight Tickets Search.story");  
    eng.run();  
}
```

Figure 59. “MyTest.java”: class for running tests on Balsamiq prototypes.

Each one of the steps in the User Story under testing makes calls to the class “MySteps.java” (flow 2) that knows which behaviors are supported. Based on the behavior referenced by the step, this class makes a call to the class “Balsamiq.java” to get the list of Balsamiq interaction elements that supports such a behavior (flow 3). The class “Balsamiq.java” in its turn makes a call to the class “MyOntology.java” (flow 4) in charge of reading the OWL file of the ontology and recovering the list of abstract interaction elements supported by a given behavior. Such a list is then returned to the class “Balsamiq.java” (flow 5) that checks, for each element returned by the ontology, which are the correspondent concrete interaction elements in Balsamiq in charge of implementing the mentioned behavior (flow 6). This mapping is recovered from the file “Balsamiq.mapping” (flow 7).

Afterward, the class “Balsamiq.java” returns such a list with the concrete Balsamiq elements to the class “MySteps.java” (flow 8) that originally made the call. With the list of supported Balsamiq elements for the step under testing, the class “MySteps.java” makes a call to the class “MyXML.java” (flow 9) in charge of parsing the Balsamiq “.bmml” file (flow 10). This parsing aims to check if the prototype carries the interaction element mentioned in the step under testing, and if so, if such an element supports the behavior mentioned in the step. The result of this parsing is then returned to the class “MySteps.java” (flow 11). At this point, based on the algorithm presented in the previous section, we verify how many instances have been found for the searched element. Finally, the class “MySteps.java” asserts the value and returns the result to the class “MyTest.java” (flow 12) that indicates if the test has failed or not.

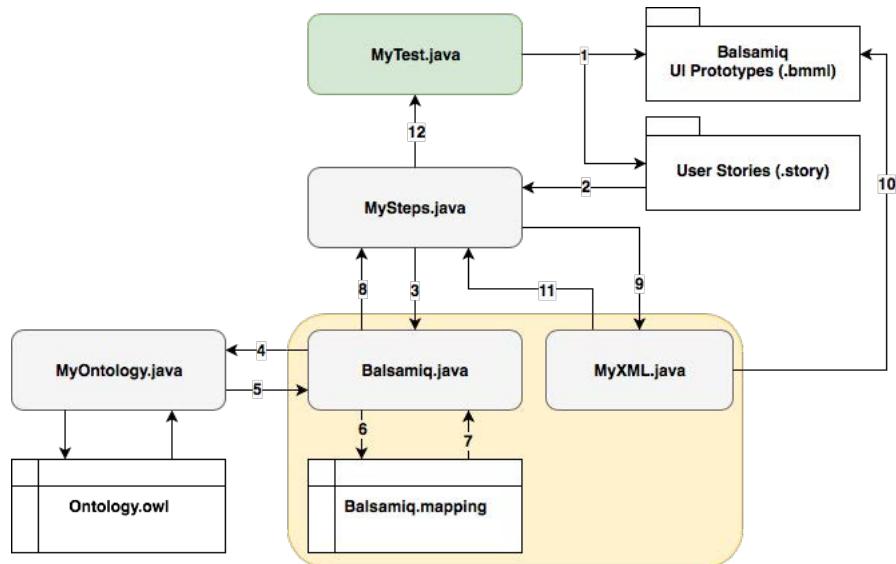


Figure 60. Flow of calls for running tests on Balsamiq prototypes.

Notice the independence of the components assigned at the core of the structure represented in Figure 60 (highlighted in yellow). Those components are related to the particularities of test implementation for Balsamiq prototypes. “Balsamiq.java” treats the demands for getting the correspondent abstract interactive elements from the ontology and translates them to the concrete interactive elements implemented by Balsamiq. “Balsamiq.mapping” provides such a translation. Finally, “MyXML.java” is in charge of parsing the BMML files of Balsamiq, searching for the element under testing. Therefore, we deliver a flexible architecture allowing, in the future, that UI prototypes modeled by other prototyping tools could also be tested by just implementing new interfaces for this core.

6.1.1.2. Setup

Considering the presented architecture, to setup and run a battery of tests, we must:

- Place the set of BMML files that will be tested in the package “Balsamiq UI Prototypes”.
- Place the set of User Stories files (“.story”) that will be tested in the package “User Stories”.
- Indicate in the “MyTest” class which prototype will be tested with which User Story (only a prototype with a User Story at a time).
- Run the “MyTest” class as a JUnit Test.

6.2. Using the Ontology to Support the Development of Consistent Prototypes

The ontology presented in chapter 4 could also be used to support presentation and behavior descriptions for prototyping tools. A prototyping environment named PANDA (Prototyping using Annotation and Decision Analysis) (Hak, Winckler and Navarre, 2016) has been developed based on this principle. The development of a prototype using this tool is made thanks to a toolbar containing widgets automatically generated from our ontology. Once the toolbar is generated, the user can create his prototype by placing widgets, whose properties are described in the ontology and presented in the edition area as illustrated in Figure 61. The use of this

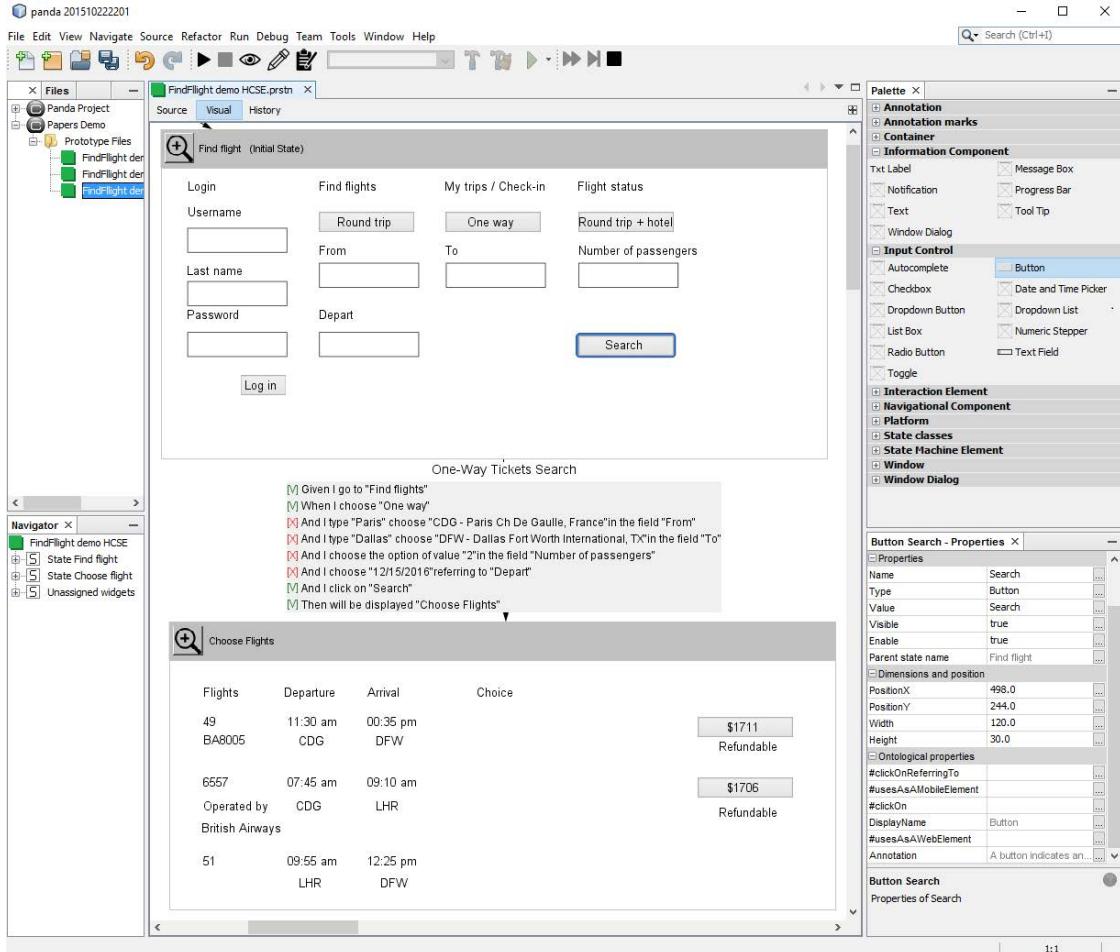


Figure 61. PANDA screenshot.

technique allows a mapping between the elements described in the ontology (and thus, their properties and supported behaviors) and each of the prototype's widgets.

A PANDA prototype features a state machine where states of the system are populated with the elements in the display when the state is active. By linking states with transitions, it is possible to specify the structure and the behavior of the prototype. After having developed the prototype, it is possible to replace a transition with a scenario. Indeed, in Figure 61 we have a testing scenario used as a transition in the state machine. This scenario links together the state “Find Flight” represented by the rectangle with a gray header in the upper part of the prototype with the state “Choose Flight” located in the lower part. The state “Find Flight” represents the initial condition (indicated by the “Given” clause) and the state “Choose Flight” represents the result of the scenario execution (indicated by the “Then” clause).

PANDA supports scenarios described in a text format which are imported in the edition area. When importing a scenario, PANDA parses the different steps and analyzes them by identifying the events, the tasks, the associated values and the targets of the task. This identification is done by splitting each line of the scenario and identifying keywords like “Given” or “Then” and the quote character. Quoted segments are interpreted as values except for the last quoted element of each line, which is identified as the target of the task. Segments before the quoted elements are considered as actions related to the values read. Each line read is then registered as a Step of the Scenario. Figure 62 shows an example for the Step “And I type ‘Paris’ and choose ‘CDG - Paris Ch De Gaulle, France’ in the field ‘From’”. The value “Paris” is associated with the action “I

type”, “CDG – Paris Ch De Gaulle, France” is associated to the action “choose” and “From” is associated with the locator “in the field”. Keywords are ignored except for the word “Given” and “Then” which introduce conditions and the final actions.

And	I type	“Paris”	and	choose	“CDG - Paris Ch De Gaulle, France”	in the field	“From”
-----	--------	---------	-----	--------	------------------------------------	--------------	--------

Figure 62. Example of a step split during its parsing.

Once the scenario has been parsed and attached between an initial and a resultant state, it can be executed in order to find out if the scenario is supported by the prototype. This execution can be made step-by-step or with the whole set of steps of the scenario being executed at the same time. PANDA checks the state in which the prototype is, as well as the properties defined in the ontology loaded. Thereby, it verifies if each step of the scenario is able to be run according to the set of supported tasks. To do so, the system starts by mapping between the widgets of the prototype and the target of the tasks during the execution, since scenarios and states of the prototype are independent. So far, this mapping is based on the name of the widget, but other mapping methods will also be considered. Then, for each step whose target has been mapped, the system checks if each action or property matches with the properties of the widget which were defined in the ontology. As an example, in the step “And I click on “Search”, PANDA looks for any widget named “Search” in the initial state and checks if the description of the corresponding widget in the ontology supports the behavior “clickOn” (see Figure 63).

The results of the tests are displayed by a colored symbol next to each step as shown in Figure 64. A red “X” represents failure, a green “V” represents success, and a black “?” represents an untested step. There is currently no distinction between the different reasons for test failure (e.g. widget not found, property not supported, etc.). In our example, the button supports the event “#clickOn” which matches with the action “I click on” of the scenario. However, none of the UI Elements (Calendar, CheckBox, Link or Radio Button) described in the ontology to support the behavior “chooseReferringTo” was found.

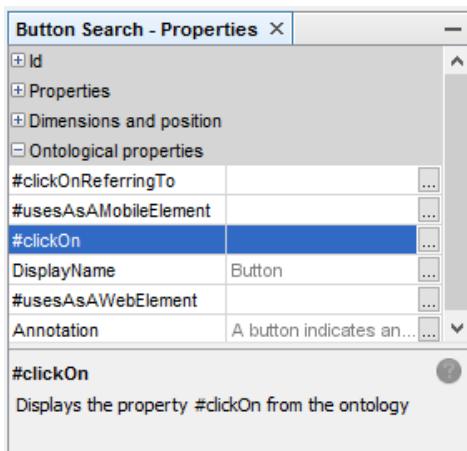


Figure 63. Properties of a button in the tool PANDA with properties defined by the ontology.

[X] And I choose "12/15/2016"referring to "Depart"
 [V] And I click on "Search"
 [?] Then will be displayed "Choose Flights"

Figure 64. Example of results given during a Scenario testing.

In a prototyping context, the automated interface testing could be used as a way to validate a version of a prototype that passes the tests or points out parts of the prototype that require attention and further analysis, for example. PANDA is focused on the evolution of a prototype, as signalized by the evolutionary cycle in the workflow shown in chapter 3. Thereby, the same

scenario can be used on different versions of the prototype, until the prototype reaches the final UI.

6.3. Evolving UI Prototypes

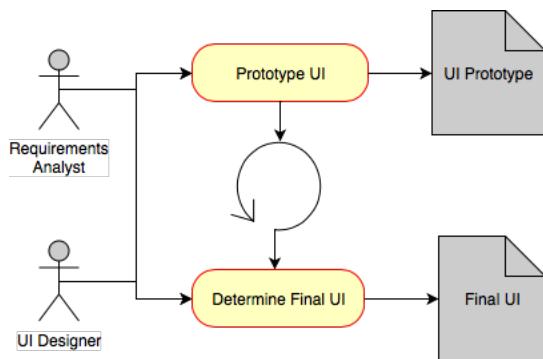


Figure 65. Activity of evolving UI prototypes.

When evolving UI prototypes (activity represented in Figure 65), we get artifacts modeling detailed or even definitive design solutions for UIs. Such evolved artifacts represent a design refinement regarding the previous solutions presented in less refined versions of UI prototypes. Thereby, it is worthy keeping the track and check the consistency of such multiple design solutions along the design process.

For the purpose of illustration concerning how our approach can support such a traceability and consistency checking, Figure 66 presents the successive mapping of a less refined Balsamiq prototype, a PANDA prototype, and a final UI when testing the User Story “Flight Tickets Search”. In the first transition, the Balsamiq prototype, designed in Figure 54 previously, is evolving to a more refined level by using PANDA. Notice that more detailed decisions about the design solution have already been taken. For example, suppose that during the project a business decision has been taken to evolve the user requirements in order to provide a new option for booking hotels along with the flights. Thereby, instead of using a simple “Round trip / One way” ButtonBar, the PANDA prototype has been modeled using a three-button solution with a third option to book hotels in addition to the round trip / one-way flight options. None of the solutions however is covered by the ontology for the behavior *I choose ... referring to ...*, so the test fails. The ButtonBar used in the Balsamiq prototype is not an interaction element modeled and recognized by the ontology, and the three-button solution used in the PANDA prototype does not allow an action of choosing, once such a kind of behavior are not supported by buttons. On the final UI, links have been chosen instead, so the test passes.

In the following example, fields like “Departure” and “Destination” became simple Text Fields once PANDA has not a specific widget for modeling the auto-complete behavior. As the step specified in the scenario is *I inform ... and choose ...*, and such a behavior is only supported by AutoComplete fields, the test fails in the PANDA prototype. For the field “Number of passengers”, the test fails as well, once the step in the User Story specifies the behavior *I choose the option of value ... in the field ...* for this interaction element, and such a behavior is not supported by the “Text Field” which has been chosen in the PANDA prototype. Notice that the PANDA prototype and, obviously, the final UI, both support the dialog description and changes the UI according to the selection made in another field. In the example, as the a “One-way” trip has been selected, then the field for specifying an arrival date is not shown. The other fields have been correctly addressed in the 3 versions of the UI. Figure 67 and Figure 68 give another example to compare the user interface refinement for performing the User Story “Choose Flights”.

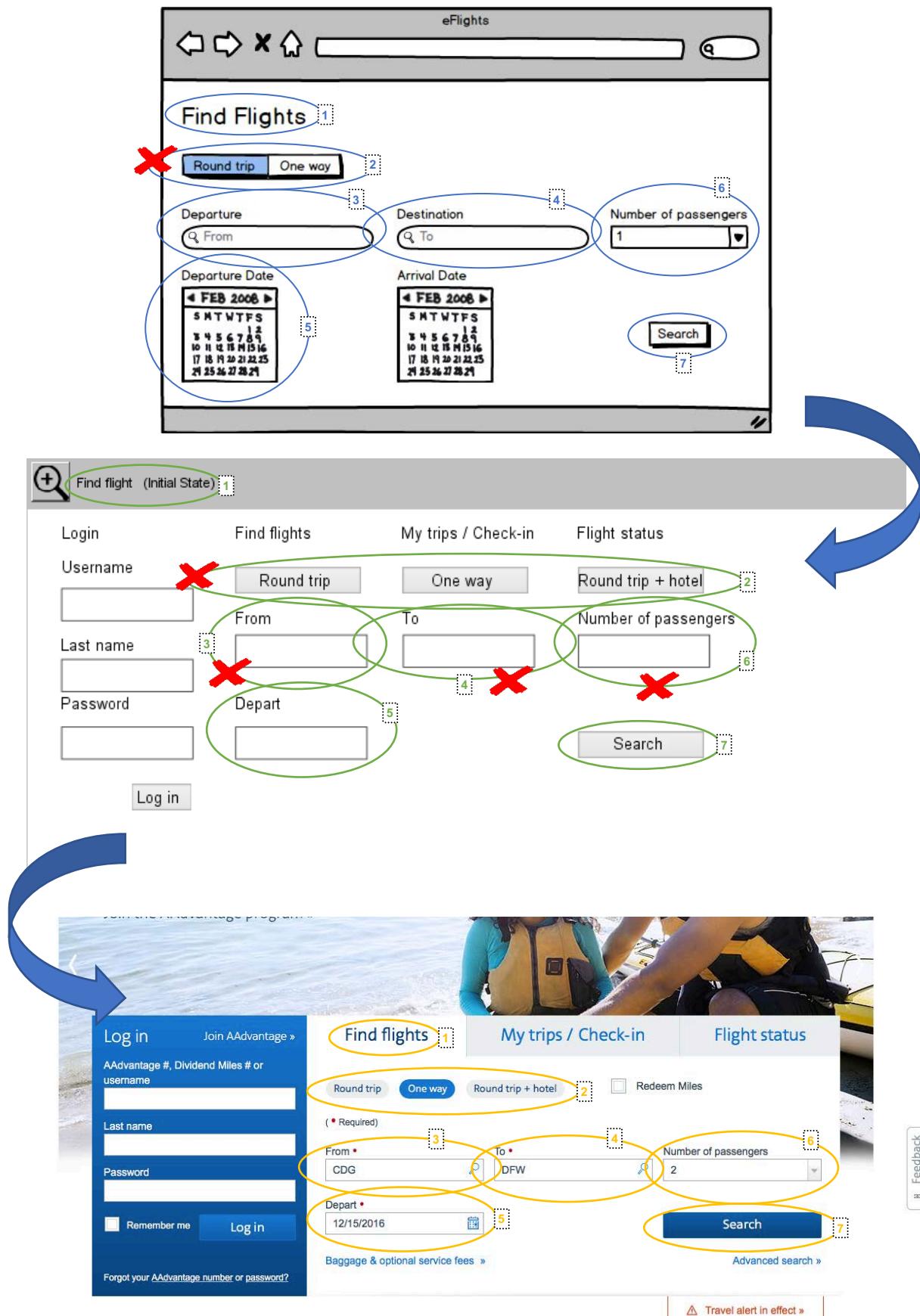


Figure 66. The less refined prototype for “Flight Tickets Search” evolving to a more refined one, and then to a final UI.

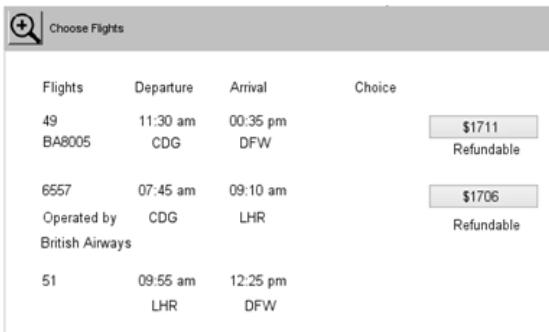


Figure 67. The “Choose Flights” UI prototype in PANDA.

The screenshot shows the American Airlines "Choose Flights" page. At the top, there are links for Home, Log In, English, Travel Information, AAAdvantage, and a search bar. Below that is a navigation bar with Find Flights, Choose Flights, Travelers, Trip Options, Select Seats, Review & Pay, and Finish. The main area is titled "Choose Flights" and shows a search form for "Choose Your Departure Flight" on Thursday, December 15, 2016. The results grid lists three flights from CDG to DFW on that date, each with a price of \$1706 and a "Refundable" status. A sidebar on the right shows a shopping cart with one item and a bonus offer for 30,000 miles.

Figure 68. The “Choose Flights” final UI.

6.3.1. Elements Mapped for Testing

The testing of UI design artifacts like UI prototypes is conducted by automatically checking whether user requirements in the User Stories have been consistently modeled in the various levels of UI refinement. Table 16 exemplifies the correspondence of concepts (model and ontology) used by our testing algorithm for the different UI instances (Balsamiq and PANDA prototypes, and final UI). In the example, the consistency of the requirements representation for the Scenario “One-Way Tickets Search” is being checked in the respective UI prototypes.

Artifact	Concepts		Step of Scenario
	Model (UI Elements)	Ontology	
Balsamiq prototype	BrowserWindow		
PANDA prototype	Browser Window	Interaction Element: Browser Window	Given I go to “Find Flights”
Final UI	Screen		
Balsamiq prototype	ButtonBar	Interaction Elements: Calendar, Checkbox, Radio Button, and Link.	When I choose “One way” referring to “Trip Type”
PANDA prototype	Button		
Final UI	Link		
Balsamiq prototype	SearchBox	Interaction Element: Autocomplete	And I inform “Departure City” and choose “Departure Airport” in the field “Departure”
PANDA prototype	Text Field		
Final UI	AutoComplete		
Balsamiq prototype	SearchBox	Interaction Element: Autocomplete	And I inform “Arrival City” and choose “Arrival Airport” in the field “Destination”
PANDA prototype	Text Field		
Final UI	AutoComplete		
Balsamiq prototype	Calendar	Interaction Elements: Dropdown List, Text Field, Autocomplete, and Calendar	When I set “Valid Departure Date” in the field “Departure Date”
PANDA prototype	Text Field		
Final UI	TextField		
Balsamiq prototype	ComboBox	Interaction Element: Dropdown List	And I choose the option of value “2” in the field “Number of passengers”
PANDA prototype	Text Field		
Final UI	Select		
Balsamiq prototype	Button	Interaction Elements: Menu, Menu Item, Button, and Link	And I click on “Search”
PANDA prototype	Button		
Final UI	Button		
Balsamiq prototype	Paragraph	Interaction Element: Text	Then will be displayed “List of Available Flights”
PANDA prototype	Text		
Final UI	Text		

Table 16. Example of concept mapping for testing.

Notice that as PANDA charges a pallet of UI elements gathered from the ontology, so in the mapping, concepts related to the model and to the ontology are exactly the same. A complete concept mapping table for all Balsamiq and final UI elements supported by the ontology is presented in the Appendix A.

6.4. Testing Final User Interfaces

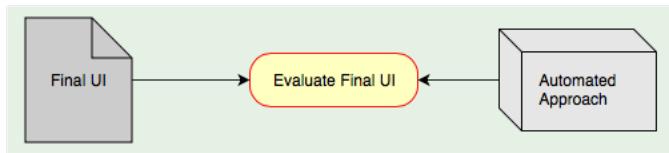


Figure 69. Activity of evaluating Final UIs.

Figure 69 reproduces the automated activity of testing final UIs. The testing of final user interfaces we present in this section involves running tests directly on the web browser of a web application. As stated in the beginning of this chapter, different testing frameworks would be

required for performing tests on different environments. Despite our ontology supports a specification for both web and mobile environments, so far, we have only explored an architecture to perform tests on a web environment, i.e. running on a web browser.

Besides using a framework to control navigation on a web browser, other frameworks are required to parse the text on User Stories, to build the test suit, or even to generate reports from the execution. To test final UIs directly from User Stories, we use external frameworks to provide automated execution on the final UI. Such frameworks are able to mimic user interactions with the final UI by running the set of scenarios described in the User Stories. Therefore, we have built an architecture of tools to bring together the multiple set of required frameworks for performing our testing approach on final user interfaces. Such an architecture is presented hereafter.

6.4.1. Integrated Tools Architecture

The integrated tools architecture we propose for testing final user interfaces is essentially based on Demoiselle Behave, JBehave, Selenium WebDriver, JUnit and Maven. We use Selenium WebDriver to run navigational behavior, and JBehave and Demoiselle Behave to parse the scenario script. Test results provided by the JUnit API indicate visually which tests are passed and which ones failed and why. Execution reports of User Stories, scenarios and steps can also be obtained by using the JBehave API.

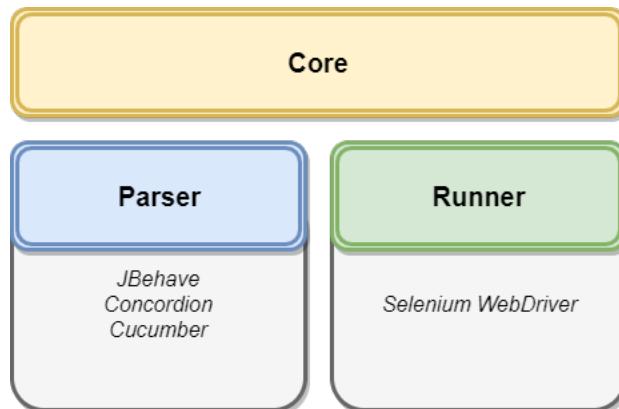


Figure 70. A 3-module integrated tools architecture.

**Figure 71.** Flow of components in the proposed architecture.

Such an architecture allows users to automate testing on web user interfaces by following our behavior-based approach. The architecture has three main modules: Core, Parser, and Runner (Figure 70). Core is responsible for the main interfaces of the framework by orchestrating the information among the other 3 modules. The Parser is responsible for the abstraction of the component that will transform the story into Java code, to send to the Runner through standard or project-specific sentences. The Runner is responsible for the abstraction of the component that will perform navigation on the user interface, such as Selenium WebDriver or even JUnit directly. The framework identifies stories written in TXT to be sent to the Parser module and later to Runner, which is responsible for interacting with a web browser using the Selenium WebDriver. Figure X illustrates such modules.

To run tests in such an architecture, story files are charged as inputs for the parser, that translates the natural language behaviors into java methods, and then selects a runner to perform the navigational commands on a given target web browser. This flow of components is illustrated in Figure 71.

6.4.2. Implementation

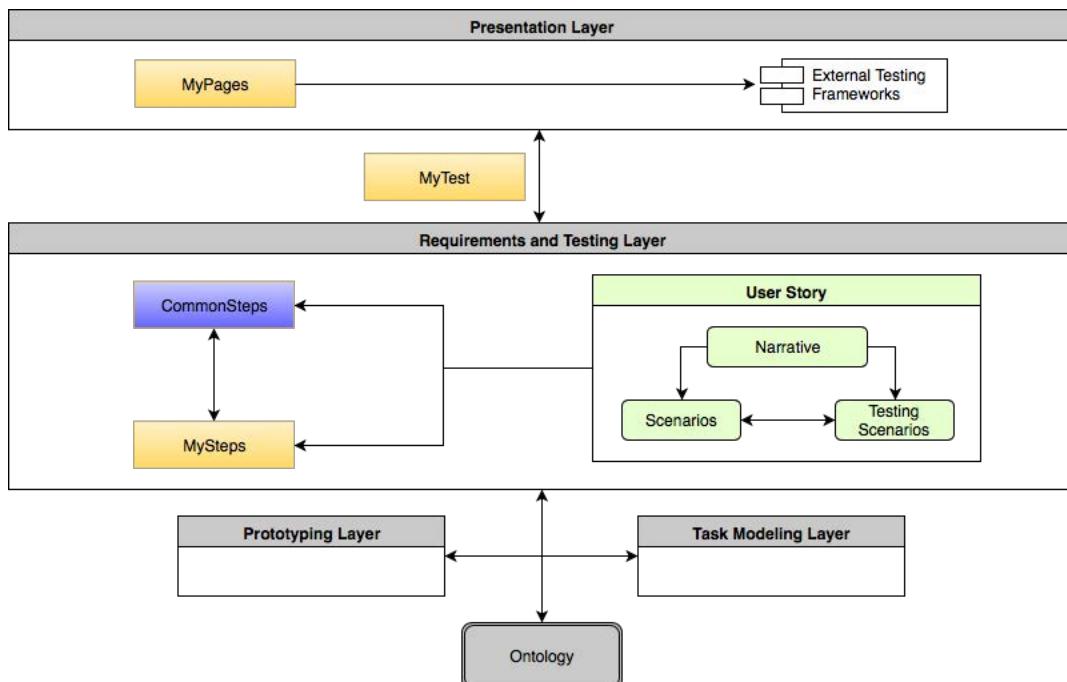
**Figure 72.** Packages and classes being structured to implement our testing approach.

Figure 72 details how we have structured packages and classes in different layers to implement our architectural approach. The ontology described in chapter 4 provides to the model a pre-defined set of behaviors used in the Requirements and Testing Layer. Artifacts produced in Prototyping and Task Modeling Layers are suitable to not only benefit from the ontology description in order to model better requirements, but also to contribute with the development of new User Stories. Pre-defined behaviors charged from the ontology are implemented by the

CommonSteps class. New extended behaviors, that are not initially covered by the ontology, can be implemented in the MySteps class. Steps in User Stories are mapped to either CommonSteps or MySteps behaviors in order to be run as Java methods. Figure 73 illustrates this mechanism.

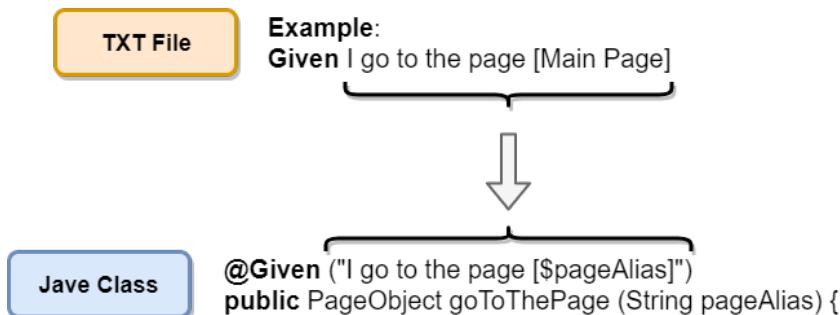


Figure 73. Parsing a step from a TXT file to a Java method.

The Presentation Layer includes the MyPages class which implements the link between abstract UI components defined in the ontology and the concrete UI components instantiated on the interface under testing. This link is crucial to allow the Selenium WebDriver and other external testing frameworks to automatically run scenarios in the right components on the UI. To link these components, the MyPages class identifies a screen map (“@ScreenMap”) which address the web page location, and several element maps (“@ElementMap”) which link the various abstract UI elements in the User Stories with their concrete UI siblings on the user interface. This link is made by manually associating the name of each abstract UI element with their concrete locators (such as IDs, XPaths, or any other web element identifier). Figure 74 illustrates this mechanism.

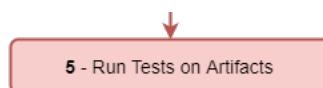
Finally, the MyTest class is a JUnit class in charge of triggering the tests, pointing which scenarios should be executed at a time, besides making the bridge between UI components in the Presentation Layer and executable behaviors in the Requirements and Testing Layer.

```

@ScreenMap(name = "Flight Search", location = "https://www.aa.com/homePage.do?locale=en_US")
public class MyPages {
    @ElementMap(name = "Round trip", locatorType = ElementLocatorType.XPATH, locator = "//*[@id='journeyTypeRT']")
    private Radio RoundTrip;
    @ElementMap(name = "One-way", locatorType = ElementLocatorType.XPATH, locator = "//*[@id='journeyTypeOW']")
    private Radio OneWay;
    @ElementMap(name = "Departure Date", locatorType = ElementLocatorType.XPATH, locator = "id('depDate')")
    private TextField DepartureDate;
}
    
```

Figure 74. MyPage Java class.

These three basic classes (MySteps, MyPages and MyTest) can also be modeled with different names into packages “steps”, “pages” and “tests”, in order to separate concerns and implements different classes for different pages or features.



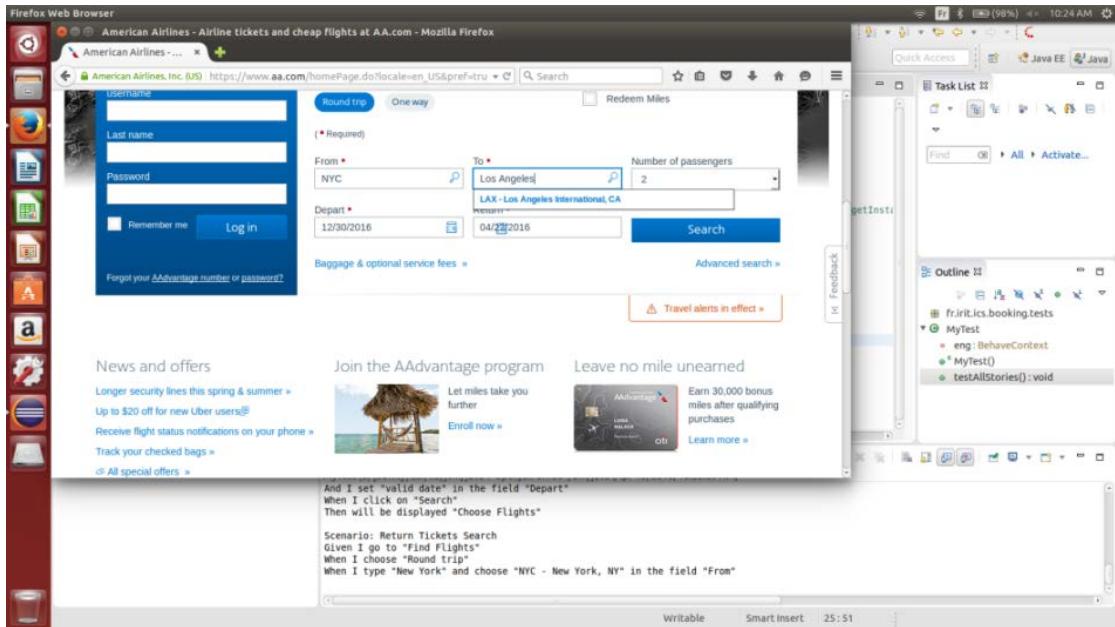


Figure 75. Automated execution of the scenario “Return Tickets Search”.

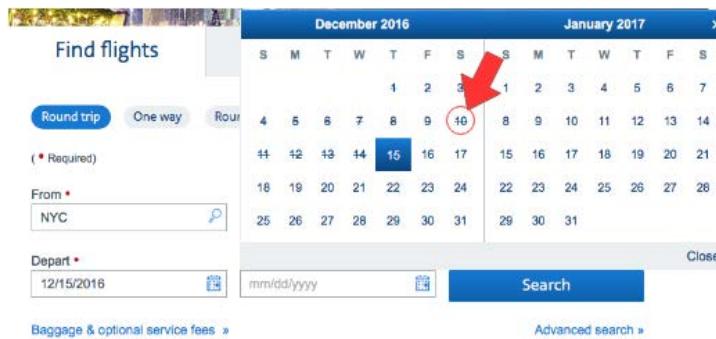


Figure 76. An attempt to select a return date before the departure date.

The environment for implementing and running the tests is the Eclipse IDE with a Maven project instantiated. Figure 75 shows such an environment with the MyTest class automatically running the Scenario “Return Tickets Search” presented in our illustrative case study. Thereafter is also presented in Figure 76 an example of test running when assessing the business rules for “Search for flights more than one year in advance” and “Search for a return flight before a departure flight”. In the example, the designers have chosen to block the inappropriate dates in the calendar according to the business rules. The figure actually shows our algorithm trying to set those invalid dates to test the rules.

The structure of the Java project is presented in Figure 77. Notice that the three aforementioned classes are packed in the package “java” and the User Stories in the package “resources”. On the right side of the figure, the structure of the MyTest class is presented highlighting the addition of the new extended behaviors in the MySteps class, and all the stories in the “/stories” folder being triggered by a JUnit test method.

A resource that facilitates the written of User Stories (and that is also available when writing and editing User Stories for assessing task models and UI prototypes) is the immediate feedback concerning the existence of behaviors in the ontology to address the step that is being written. Figure 78 illustrates this resource. Notice that all the steps in the scenario have been recognized,

**Figure 77.** Package tree (on the left) and MyTest class (on the right).

i.e. there are equivalent behaviors in the ontology to address them, except the step “When I set the date ‘12/20/2017’ in the field ‘Return’” that has been underlined to alert that such a step is not recognized by the ontology (actually the right step in this case is generic: “... I set ‘<value>’ in the field ‘<element>’”, like has been used in the following line). When clicking in the alert icon, a message to say that “no step is matching” will be shown. Additional feedback is also given recognizing in the step the mention to values and interactive elements when they are surrounded by quotation marks.

**Figure 78.** Writing a User Story and getting instant feedback of unknown steps.

The testing results are presented through the classical JUnit green/red bar within the Eclipse IDE. By the end of the tests, a JBehave detailed report is automatically generated in the project folder. Additionally, for each error found, screenshots are taken and stored to allow a better analyze of the results afterwards. Examples of these features are presented in Figure 79.

The image shows two screenshots of the Eclipse IDE. The top screenshot shows the Project Explorer with a green bar indicating 'Runs: 1/1 Errors: 0 Failures: 0' and 'Finished after 2.392 seconds'. The bottom screenshot shows the Project Explorer with a red bar indicating 'Runs: 1/1 Errors: 0 Failures: 1' and 'Finished after 47.896 seconds'. Below the Project Explorer is the JBehave 'Story Reports' table.

Stories	Scenarios										GivenStory Scenarios					Steps			Not Performed	Ignorable	Duration (hh:mm:ss.SSS)	View
	Name	Excluded	Total	Successful	Pending	Failed	Excluded	Total	Successful	Pending	Failed	Excluded	Total	Successful	Pending	Not Performed	Ignorable	Duration (hh:mm:ss.SSS)	View			
Access	0	2	2	0	0	0	0	0	0	0	0	9	9	0	0	0	0	00:00:00.000	stats.xml html			
AfterStories	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00:00:00.000	stats.xml html			
BeforeStories	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	00:00:00.000	stats.xml html			
Choice	0	1	0	0	1	0	0	0	0	0	0	21	18	0	1	2	0	00:00:00.000	stats.xml html			
Confirm Flight Selection	0	2	2	2	0	0	0	0	0	0	0	26	23	2	0	1	0	00:00:00.000	stats.xml html			
Debehave	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	00:00:00.000	stats.xml html			
Flight	0	1	1	0	0	0	0	0	0	0	0	18	18	0	0	0	0	00:00:00.000	stats.xml html			
Flight Tickets Search	0	5	5	0	0	0	0	0	0	0	0	72	72	0	0	0	0	00:00:00.000	stats.xml html			
Search	0	1	1	0	0	0	0	0	0	0	0	17	17	0	0	0	0	00:00:00.000	stats.xml html			
Select A Suitable Flight	0	1	1	0	0	0	0	0	0	0	0	26	26	0	0	0	0	00:00:00.000	stats.xml html			
Test	0	2	2	0	0	0	0	0	0	0	0	17	4	4	0	9	0	00:00:00.000	stats.xml html			
Voo	0	1	1	0	0	0	0	0	0	0	0	18	18	0	0	0	0	00:00:00.000	stats.xml html			
	12	0	17	16	5	1	0	0	0	0	0	224	205	6	1	12	0	00:00:46.000	Totals			

Figure 79. JUnit green/red bar at the left, and JBehave detailed report at the right.

6.4.3. Handling Test Data

Test data are an important component of software testing. They are very useful providing concrete examples for scenarios, but they must be carefully planned to explore the multiple black-

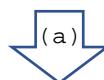
box test design techniques such as equivalence partitioning, boundary value analysis, domain analysis and so on. Test data also become out of date easily, especially those ones that reference time variables such as dates, ages, etc. Providing strategies for organizing and maintaining such test data is therefore crucial for getting well-succeed test scenarios.

In our approach test data can be specified directly in the step of User Stories or be specified as data domains (variables) in the step, keeping the real test data out of the scenario. Our approach offers two main strategies to set test data out of scenarios. The first one is to use Data Providers to store values for variables which can be used to write the steps of scenarios. Data Providers associate such variables (specified in the step) to the real test data (specified in the test source code) in a Java method directly in the source code. The real test data is then injected into the scenario at runtime. This mechanism is useful to render flexible the reuse of data dynamically and to hide data in scenarios without losing readability. The downside is that data are encapsulated in the source code which harms their maintainability. Figure 80 illustrates this mechanism.

The second mechanism is the use of data storages in XML files. With data stored in XML files, the test source code is kept clean and the maintainability of test data is considerably improved. Unlike Data Providers, data storages associate the variables (specified in the step) to the real test data by using steps of scenarios referencing specific behaviors provided by the ontology. Figure 81 illustrates this mechanism. In the example, data stored for the variables “Number of passengers” and “Depart” are associated respectively to the values “2” and “12/15/2016” for a “Europe USA” trip scenario, and to the values “3” and “12/31/2016” for a “Inside USA” trip scenario. At runtime, these real test data are assigned to the respective steps (transition “a” in Figure 81) and then used in the respective scenarios of searching flights “Europe USA” and “Inside USA” (transition “b” in Figure 81). This mechanism is useful to work with a large set of data that should be introduced in scenarios at runtime. The downside is that scenarios can eventually lose readability due to the multiple references to other steps of other scenarios which indeed get the data from the storages.

Specific interactive behaviors for manipulating data providers and data storages are classified in the ontology as “Data Provider Behaviors” (Table 8) in chapter 4.

```
dataProvider.put("valid date", "12/30/2016");
```



And I choose "valid date" referring to "Depart" ...

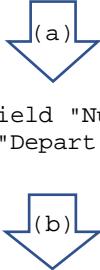
Figure 80. Data in Data Provider: (a) data being associated to a variable to be used in the step.

```
<DataSet>
  <dataRecords>
    <DataRecord id="Europe USA">
      <dataItems>
        <DataItem key="Number of passengers" value="2" />
        <DataItem key="Depart" value="12/15/2016" />
      </dataItems>
    </DataRecord>
    <DataRecord id="Inside USA">
      <dataItems>
        <DataItem key="Number of passengers" value="3" />
```

```

<DataItem key="Depart" value="12/31/2016" />
...
</DataSet>

```



... When I provide the value of the field "Number of passengers"
And I provide the value of the field "Depart" ...

Scenario: Search of flights stored in the dataset
When I search for flights "Europe USA"
Then "Choose Flights" is displayed
When I search for flights "Inside USA"
Then "Choose Flights" is displayed

Figure 81. Data stored in an XML file: (a) data associated to XML file, (b) reference to dataset.

6.5. Conclusion

The approach we describe in this chapter for assessing user interface prototypes has the main advantage of ensuring a reliable correspondence between different interaction elements modeled in prototypes with different levels of refinement. By using an ontology to support a wide description of interaction elements and their related behaviors when subject to a user interaction, this approach succeeds to provide automated testing for Balsamiq prototypes, as well as for final UIs developed under whatever technology for designing the presentation layer on web pages. Additionally, the ontology can be used as a base specification for developing new prototyping tools like PANDA, which will be able to produce UI prototypes already consistent with a large set of user-system interactive behaviors.

For implementing this approach, we have also proposed an open and flexible architecture where different approaches and tools for designing UI prototypes could fit in the future. For prototypes with a low level of refinement, it is enough to implement a new core interface for describing the way such tools deal with their interaction elements and how they can be identified in their source files. For final UIs, it is enough to replace Selenium WebDriver by another testing framework adapted to running tests on user interfaces in other environments such as mobile and desktop.

In this chapter we make use of both static and co-execution strategies for assessing user interface prototypes depending on their level of refinement. Like for task models, when opting for a static analysis of Balsamiq source files, we gain in performance and availability of tests. Specially in environments requiring a high-availability of tests to be executed continuously along multiple iterations, static approaches benefit from an instantaneous consistency checking analyzing several hundreds of source files at the same time.

For final UIs, we implemented the strategy of co-execution. Co-execution approaches have the benefit of allowing running models simultaneously with a visual feedback at real-time about the correspondence of entities that are being assessed in each model. As we stated before, such approaches usually have the drawback of demanding a high investment to prepare and adapt the artifacts for testing. In our approach however, such an investment is restricted to the mapping of interactive elements on the respective user interfaces under testing. As the great benefit of co-execution on final UIs is providing a visual feedback during the execution simulating a real user

interacting with interaction elements at real-time, this process can end up being very slow with the growing number of user interfaces and scenarios to be tested. As far as the simulation of real user actions is not a concern, such a drawback can be reduced by using GUI-less browser implementations such as HtmlUnit⁵, which benefits from high-level manipulation of web pages without the need of bringing the browser to the front and co-executing the simulated user actions. Like static approaches, this strategy is suitable for environments demanding high-availability and continuous testing.

Finally, with both strategies for running automated tests on UI prototypes together with our static strategy for assessing task models, we set out a step forward within the process of fully automating software verification, validation and testing (VV&T). As an integrated approach, the same set of User Stories is assigned to automatically assess both task models and UI prototypes in different levels of abstraction, ensuring a consistent VV&T approach for interactive systems with high-availability of instantaneous feedback about the consistency of artifacts with the user requirements.

Further in this thesis, the chapter 8 will demonstrate how this approach performed when applied to a large case study, and how the UI prototypes we produced have been ensured as consistent with other user interface design artifacts like task models. The chapter also details a broad set of inconsistencies our approach is able to identify and provides results about its potential.

6.6. Resultant Publications

Silva, T. R. & Winckler, M. (2017). A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts. In: Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems (IHC 2017), pp. 21-30. ACM. DOI: <http://doi.org/10.1145/3160504.3160506>. (Silva and Winckler, 2017)

Silva, T. R., Hak, J. L. & Winckler, M. (2016). Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development. In: 6th International Working Conference on Human-Centred Software Engineering, and 8th International Working Conference on Human Error, Safety, and System Development (HCSE 2016 and HESSD 2016), pp. 86-107, vol. 9856. Lecture Notes in Computer Science, Springer International Publishing. DOI: http://doi.org/10.1007/978-3-319-44902-9_7. (Silva, Hak and Winckler, 2016b)

Silva, T. R., Hak, J. L. & Winckler, M. (2016). An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development. Complex Systems Informatics and Modeling Quarterly, 1 (7), pp. 81-107. DOI: <http://doi.org/10.7250/csimg.2016-7.05>. (Silva, Hak and Winckler, 2016a)

⁵ <http://htmlunit.sourceforge.net>

Part III - Evaluation

Chapter 7

Case Study 1 - Understandability of User Stories

Summary

This chapter presents the experimental design and the results of our first case study to evaluate the understandability, by potential Product Owners, of the User Stories template which we have used to describe user requirements in our approach. For that, it has been chosen the department in charge of business trips in our institute. The experiment has been conducted with 4 members of the team in one-hour sections of interviews. During this time, it has been captured their user impressions about the current system support for booking business travels and how it could be better. Based on that, the participants were invited to write their own User Stories to describe a feature they are used to perform. The results have been used to analyze their understandability of User Stories structure and the adherence of such stories to the ontological pattern we defined for our approach.

To present our finding, this chapter is divided in 7 sections. The first one (section 7.1) presents our experimental design, detailing our research questions and measures we used to assess the outcomes. Following this, we present the business narrative to give the context of how business travels are booked in our institute (section 7.2). Next, we detail our methodology to conduct the study (section 7.3), followed by the participant's profile (section 7.4), and the exercise we proposed to allow them writing their own User Stories (section 7.5). Section 7.6 brings the results of the study, highlighting the set of User Stories written by the participants, our adherence analyses considering stories and scenarios, our discussion of such results, our general findings and implications, and the threats to validity of this study. Finally, section 7.7 concludes our remarks and points out future investigation opportunities in this field.

The travel department from our research institute (Toulouse Institute of Computer Science Research - IRIT) has been selected as a target group to conduct the present case study. This choice has been made because the travel department team is in the target population of stakeholders in our approach. They receive multiple and varied demands of business trips to follow and validate, which come from the whole team of researchers at the institute. Demands are likely to bring difficulties and problems experienced by researchers when trying to book their business trips directly through internal systems, which has a huge potential to bring prospective features to be developed or improved in such systems. By acting as such a hub, the participants from the travel department considered in this study act actually as Product Owners (POs) (Schwaber, 2004), once they master the current business process and have the potential to integrate a specialized group for eventually specifying requirements to maintain or develop a new software system in this business field.

7.1. Experimental Design

The present case study has been designed around two research questions:

RQ 1. Are participants able to read/understand a basic User Story template and use it to write their own stories?

To answer this research question, we measure the adherence of the User Stories produced by the participants to the structure of the template initially presented to them. This adherence is measured following the scale presented below.

RQ 2. Which is the vocabulary the participants make use when writing their own User Stories?

To answer this research question, we measure the adherence of the vocabulary used by the participants when writing their User Stories to the predefined interactive behaviors modeled in the ontology. This adherence is measured following the scale presented below.

The adherence analysis has been made separately for the first part of the User Story (narrative section) and for the related scenario (scenario section), observing the existent gap between the steps each participant specified and the equivalent and available steps in the ontology. For each statement in the User Story, we have classified its adherence to the template or to the ontology by using the following scale:

- **Null** adherence – scale 0 ○○○○○
- **Very Low** adherence – scale 1 ●○○○○○
- **Low** adherence – scale 2 ●●○○○○
- **Medium** adherence – scale 3 ●●●○○○
- **High** adherence – scale 4 ●●●●○○
- **Very High** adherence – scale 5 ●●●●●○
- **Full** adherence – scale 6 ●●●●●●

The experiment has been organized around interview and exercise sections with each one of the participants. These sections were structured in steps as follows:

- A first step aimed at capturing the profile of the participants and their impressions about the current software system.
- A second step to present and exemplify the structure of a User Story to the participants (but not the ontology).
- And a third step asking them to write their own stories.

7.2. Methodology

The study has been conducted with the group of participants along 2 weeks in May 2017. The participants were selected by their availability and heterogeneity of profiles. In total, 4 (four) participants have participated to the study. Each one of them were interviewed by us for about 1 hour. The interview conducted with each participant had three distinguished components. The first part was aimed to identify the participants' profile and their experience working with business trips. The second part was aimed to collect information concerning their impressions about the current in-use systems at the travel department. Finally, the third and last part was aimed to conduct the exercise that allowed us to observe their ability in writing the intended User Stories.

Before each session of interviews, participants were required to sign a disclosure agreement stating the exclusively use of the data for researching purposes and that their identities and personal opinions would not be used individually under any circumstances. With the agreement

of participants, all the interviews were fully audio recorded. Thus, after the beginning of each session, in the first section of the interview, the participants were invited to answer a set of 6 (six) questions about their profile. The questions covered information about their gender, age, education, for how long they were involved with that job in the travel department at our institute, whether they were previous experiences in that kind of job before joining the department, and finally a general and open question about an overview of their job and daily activities in the department. The details of this part are described in the next section “Participant’s Profile” hereafter.

In the second part of the interview, we were interested in collecting participants’ impressions about the current intern system used for booking the business trips. A total of 16 questions have been made at this second part of the interview concerning both factual and interpretation points. They were asked about how booking demands are processed and treated along a life cycle in the travel department, and about their personal opinion about constraints and improvement opportunities in the current workflow, as well as in the current in-use system they use daily for processing the booking requests.

In the third and last part of the interview, the participants received an example of User Story with a brief explanation about its general goals, structure and a single example in the context of business trips. In the sequence they were asked to produce their own User Stories for describing a feature they have faced recently when using the current software system for booking the business trips. The details of this part are described in the section “The Proposed Exercise” hereafter.

7.3. The Business Narrative

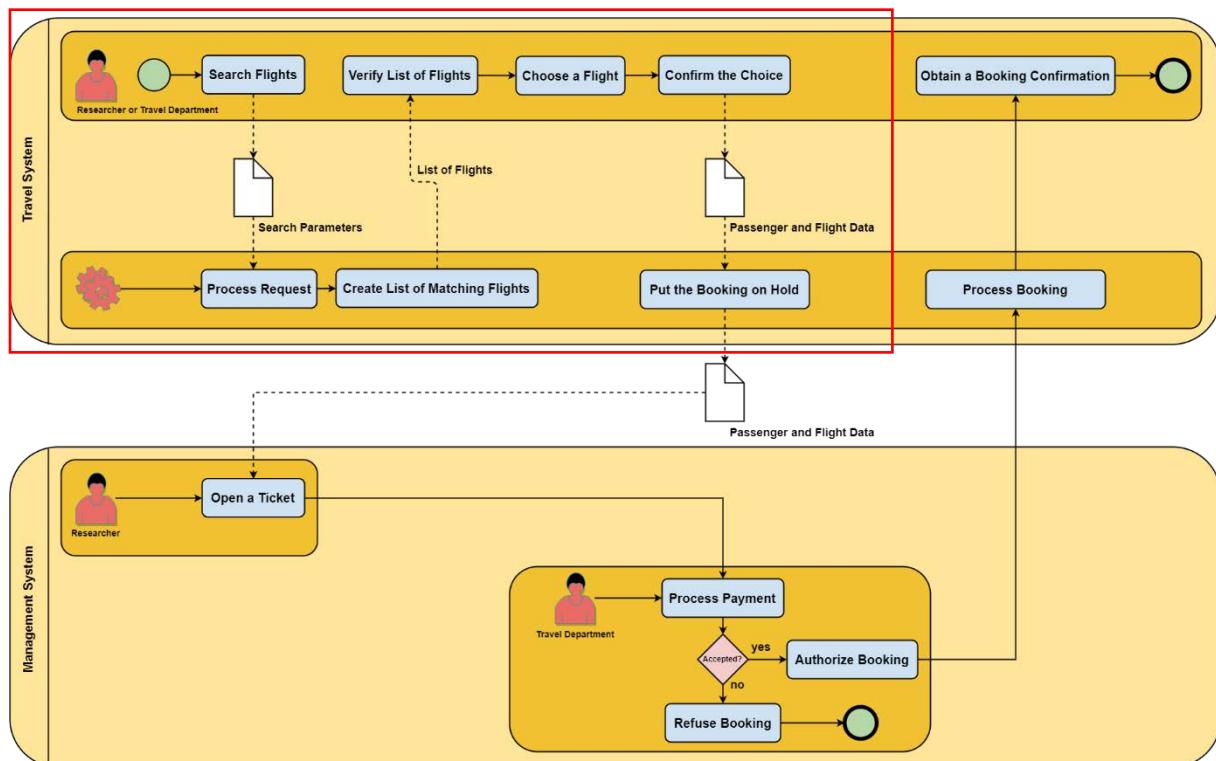


Figure 82. BPMN model for the case study.

The process of booking business trips for researchers in our department is supported by two information systems. The first one is named Travel Planet and is used by researchers for

searching their flights and getting a quotation of rates for a given itinerary. The second one is named GLPI and is used for managing demands of services for different departments by approving or declining travel quotations based on the budget available for each project or researcher. Both systems are currently in operation. In this case study, we are focused on the services demanded by researchers to the travel department related to the process of booking business trips, so our focus falls on the Travel Planet system. Figure 83 presents a screenshot of this system which both researchers and the department team have access.

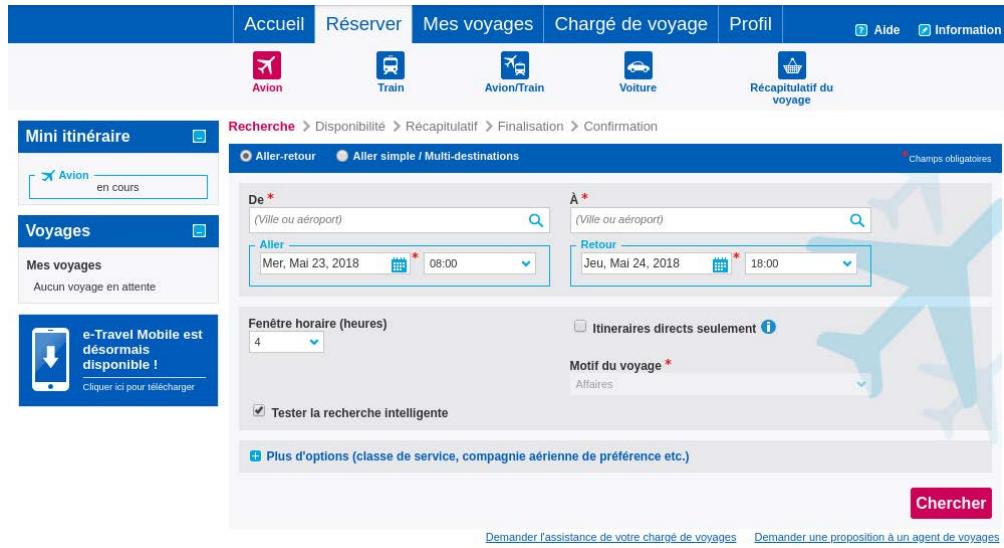


Figure 83. Travel Planet system for booking business trips.

The overall process of booking was described and detailed during the cycle of interviews with the participants and is illustrated in Figure 82 through a BPMN model. The researcher starts the process by conducting a search of flights based on a given set of parameters (such as departure and arrival cities or airports, date of departure and return, timeframe, etc.). Such search parameters are processed by the system that creates a list of matching flights, returning it to the user. The researcher then verifies the list of available flights and makes his/her choice. When he/she confirms his/her choice, passenger and flight data are saved by the system and the booking is put on hold. At this point, the researcher needs to open a ticket in the management system in order to formalize the demand of payment for the travel department. When the ticket is open, the travel department team process the payment, checking whether the research has enough budget for the trip. If the budget is enough, the payment is accepted, and the travel department team authorizes the booking. If not, the booking is refused, and the process is ended. For approved trips, the travel system finally processes the booking and the researcher receives his/her electronic ticket as a result of the booking confirmation.

For this case study, we will work on the travel system's sub process (circled in red in Figure 82). As such, the participants were invited to produce some examples of User Stories related to a feature they consider important to the system. The goal of this exercise is to get reasonably-formatted requirements from critical stakeholders in the travel department. Such requirements have been used to identify examples of use they consider relevant and to build a consistent set of requirements for assessing travel system's development artifacts. This exercise had also as objective to evaluate the level of adherence of our pre-defined behaviors specified in the ontology and their understanding and acceptance by non-technical people for writing their own requirements specification through User Stories.

7.4. Participant's Profile

The first participant (P1) was a woman, 50 year's old, secondary school level plus a complementary year of study, with 10 years of experience managing business trips, being 4 years in the targeted travel department. P1 has informed his/her daily tasks are based on managing travel demands from researches besides doing research of flights and rates for guests visiting the institute.

The second participant (P2) was a woman, 30 year's old, secondary school level plus two complementary years of study, with 6 years of experience managing business trips, being 3 years in the targeted travel department. P2 has informed his/her daily tasks are based on managing the budget of researchers, the different aspects of their business trips like housing, flight tickets, billing cycles, etc. but his/her main activity is to manage their trips.

The third participant (P3) was a woman, 52 year's old, secondary school level, with 4 years of experience managing business trips, all of them in the targeted travel department. P3 has informed his/her daily tasks are based on managing the demands, check their correctness, besides open tickets and make quotations for the trips.

Finally, the fourth and last participant (P4) was a man, 25 year's old, secondary school level, with 4 years of experience managing business trips, being just 1 month in the targeted travel department. P4 has informed his/her daily tasks are based on requests for processing the trips, booking them through the system and manage the billings.

Table 17 summarizes the participant's profile. We notice therein a homogeneity in their level of education, with P3 and P4 having completed only the secondary level, while P1 has completed one year more of undergraduate studies, and P2 two years more. We notice as well P1 is the most experienced participant with almost twice the experience of the other 3 participants. Although P1, P2 and P3 have also the same level of seniority at this charge in the institute (about 4 years in average), P4 had been hired only 1 month prior to this study, so his/her participation was interesting to compare his/her view with possible work habits acquired by the older employees. Finally, we had a predominance of women (3 of 4 participants) with a good range of ages, from 25 to 52.

Participant	Gender	Age	Education	Experience (Years in total)	Experience (Years in the institute)
P1	Female	50	SSL+1	10	4
P2	Female	30	SSL+2	6	3
P3	Female	52	SSL	4	4
P4	Man	25	SSL	4	1 month

Table 17. Participant's Profile.

7.5. The Proposed Exercise

For the proposed exercise, aim of this study, participants were invited to write manually one single User Story with one single scenario for describing a feature they have faced recently when using the current software system for booking the business trips. This activity has taken about the last twenty minutes of each interview. To do the exercise, participants were introduced to the structure and to the main components of a typical User Story based on the extended format proposed by North (North, 2017) which is the same used by our approach. Then, an example of

User Story describing a searching feature of a one-way flight for a general business trip has been presented. As the participants were French native speakers and had no English proficiency, the story provided as example and the stories produced by the participants were all written in French, and then translated to English by us.

For this exercise, apart from the short beginning explanation about the structure of a typical User Story and the single example we provided, we decided to not give any prior training to the participants. As such, we did not mention the existence of common and predefined interactive behaviors in the ontology which were supposed to be used for writing the stories, although the example of User Story we provided to them had been written following such behaviors presented in the ontology. This decision was made because one of the goals of this study was to investigate the ability of non-technical core POs to specify their own User Stories and in which extent the interactive behaviors described in the ontology would be perceived as useful enough to be spontaneously reproduced by the participants.

Herein, Figure 84 and Figure 85 are respectively the translated/equivalent version (in English) of the User Story structure and the example of a User Story we presented to the participants:

```
Title (one line describing the story)

Narrative:
As a [role]
I want [feature]
So that [benefit]

Scenario 1: Title
Given [context]
And [some more context] ...
When [event]
Then [outcome]
And [another outcome] ...

Scenario 2: ...
```

Figure 84. Structure of a User Story presented to the participants translated to English.

```
Title: Flight Tickets Search

Narrative:
As a frequent traveler
I want to be able to search tickets, providing locations and dates
So that I can obtain information about rates and times of the flights.

Scenario: One-Way Tickets Search
Given I go to the page "Find flights"
When I choose "One way"
And I type "Paris" and choose "Paris, Charles de Gaulle (CDG)" in the field "From"
And I type "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "To"
And I choose "2" in the field "Total number of passengers"
And I choose "12/15/2016" in the field "Depart"
And I click on "Search"
Then will be displayed the list of available flights
```

Figure 85. Example of a User Story presented to the participants translated to English.

7.6. Results

In the first and second parts of the interview, participants highlighted they manage about 400 travel demands per year, being a means of 12 per week in off-peak seasons, and 12 per day in

peak seasons. They are in general pretty satisfied about the current system's support, nonetheless they frequently need to contact the researchers asking for complementary information about the trip. As far as new features are a concern, a participant pointed out that having a list of departure times for the less expensive flight rates could be very interesting. Another participant pointed out the need of a feature to book several trips for a group in the same demand. They almost unanimously pointed out that features for searching multi-destination trips have certainly a room for improvement.

In the third and last part of the interview, focus of this study, we captured the User Stories written by the participants. They are detailed hereafter.

7.6.1. User Stories Writing

Preamble .

1

En tant que : Invité .

J' veux : Des billets d'avion avec horaires et vols définis .

Afin de : Réserver des billets .

Scénario : Recherche du billet demandé .

2

Etant donné : Je suis sur le site. (SIMBAB/TRAVEL)

Quand je choisis le vol demandé

~~ET~~ (destination et horaires)
Et : Je choisi type de voyageur - "invités"
~~Thalys / PARIS~~

~~ET~~ Départ 7H00 : Recherche même pour Athènes
~~ATEURS~~ → Plusieurs propositions

~~ET~~ On choisit le voyageur et

→ Quand : On Je sais b. J'unes concernant
le voyageur - (nom, prénom, date naissance
Téléphone, email),
eventuellement les cartes & Réductions
(- Flying blue et Carte abonnement).

3

~~ATEURS~~ - Billet en demande de validation .

<p>Narrative: As a guest I want airline tickets with defined time and flights So that I can book tickets</p>	1
<p>Scenario: Searching demanded tickets Given I go to the site SIMBAB/TRAVEL When I choose the demanded flight (destination and times, TOULOUSE/PARIS, departure 7 a.m., return 7 p.m. at the same day) And I choose type of traveler "Guest" And I search Then several propositions And I choose the desired flights</p>	2
<p>When I inform the data concerning the traveler (name, given name, birthdate, phone, mail), and eventually the loyalty card (Flying Blue and Season Ticket) Then ticket waiting for validation</p>	3

Figure 86. User Story written by P1.

<p>As a frequent traveler I want to search for tickets, providing locations and dates for a multi-destination trip So that I can obtain information about rates and flight times</p>
<p>Scenario: Multi-destination searching Given I go to the page "Searching Flights" When I choose "Multi-destinations" And I type "Paris" and choose "Paris, Charles de Gaulle" in the field "Departure" And I type "Rio de Janeiro" in the field "Destination" And I choose "15/02/17" in the field "Departure Date" And I choose "20/02/17" in the field "Return Date" And I type "Rio de Janeiro" in the field "Departure" And I type "Porto Alegre" in the field "Destination" And I choose "17/02/17" in the field "Departure Date" And I choose "19/02/17" in the field "Return Date" And I click on "Search" Then will be displayed the list of available flights</p>

Figure 87. User Story written by P2.

<p>As a travel manager I want to check travel authorizations So that I can ensure the confirmed bookings</p>
<p>Scenario: Listing travel authorizations Given I go to the tab "Travel Authorization" When I type the "Booking Reference" And I check if the request is well registered Then at this time, I can know for sure (or not) the request has been taken into account And it's shown a tab: authorized / non-authorized</p>

Figure 88. User Story written by P3.

<p>As an intern I want to book a flight to Paris departing on May 2nd until May 10th So that I can attend a seminar</p>
<p>Scenario: Given I'm going to book my flight When I provide all the information</p>

```
And I choose search by fares
Then all the available flights for the date are classified by ascending order of
fares.
```

Figure 89. User Story written by P4.

Figure 86, Figure 87, Figure 88, and Figure 89 present the translated versions of the User Stories written by the participants in their first attempts. Figure 86 brings additionally an example of the User Story handwritten by the participant P1.

The participant P1 (Figure 86) presents a User Story to describe the process of booking trips for a guest, i.e. an external person, normally a researcher from outside of the institute. We notice clearly that the first participant has chosen to describe the US in a high level, free of format, not necessarily paying attention to the ontology pattern step presented in the example. Thus, each step of scenarios could be identified as domain-dependent behaviors, i.e. behaviors that make direct reference to jargons used for booking flights. In the first identified point (1 in Figure 86), we can see the user states a narrative concerning a guest searching for airline tickets with defined time and flights in order to book tickets. Here we notice as well, the user has committed a mistake when identifying the role that benefits from the story. In fact, he/she identified that the guest would be the right role for this story when indeed the account managers of the travel department would be the beneficiaries, once it is them that would perform the booking using the system on behalf of the guest.

In the second identified point (2 in Figure 86), we notice the first scenario he/she identified. The scenario specifies the use of two intern systems for booking business flights. It simulates in a high-level a travel from Toulouse to Paris departing at 7 a.m. and returning at 7 p.m. at the same day. So, he/she informs this trip concerns a guest and based on the submitted search, he/she chooses the desired flights. At this point (3 in Figure 86), our user mixed a second scenario with the first one. He/she continues specifying actions for informing traveler's data and putting the ticket on hold, waiting for validation.

The participant P2 (Figure 87) reported a story for booking a multi-destination trip. We notice here that, unlike the first one, the second participant has chosen to describe the User Story closely paying attention to the ontology pattern step presented in the example. Thus, each step of the scenario could be identified as domain-independent behaviors, i.e. behaviors that refers to the actions on the user interface, without mentioning jargons used for booking flights. In the first identified point (1 in Figure 87), we can see the user states a narrative concerning a frequent traveler searching for a multi-destination ticket in order to obtain rates and flight times. In the second identified point (2 in Figure 87), a scenario for searching return flights from Paris to Rio de Janeiro with a stopover in Porto Alegre is presented. We can see the user clearly understood the structure of the scenario, once he/she adjusted the sequence of steps to cover a multi-trip data entrance with different cities and dates.

The participant P3 (Figure 88) reported a story for checking travel authorizations. In such a story, a travel manager checks travel authorizations in order to ensure that a given booking has been effectively taken into account. For that, a scenario for listing travel authorizations is specified. Therein, once the user goes to the tab "Travel Authorization", types the booking reference and checks if the request is well registered, then, according to him/her, at this time, he/she is able to ensure whether the request has been taken into account or not. The resultant behavior of the system is to show a tab with a message signalizing that the booking is authorized or non-authorized. For this user, we noticed a medium-level of adherence and understandability of the language patterns defined in the ontology.

The fourth and last participant P4 (Figure 89) reported a story in the role of a travel department's intern. He/she describes a research of flights to Paris for attending a seminar from 2nd until 10th May. The participant however has mistakenly informed data details for a specific scenario in the narrative section of the story. As a consequence, when specifying a scenario for this story, he/she supposedly makes reference to the data already informed previously in the wrong section ("When I provide all the information"). The scenario also features a search of flights classified by their ascending order of fares. We can notice, in general, the participant had difficulties to understand the structure of the stories. It makes his/her User Story hardly adherent to the implicit proposed language patterns.

Considering the seven levels of Nielsen's linguistic model of interaction (Nielsen, 1986), the stories produced by the participants contain elements that could be classified from the level 1(goal) until the level 5 (lexical).

7.6.2. Adherence Analyses

We have categorized each deviation from the proposed template committed by the participants when writing their User Stories. Such categories have been defined as adherence problems and have been classified under the Meyer's seven sins (Meyer, 1985). They are described as follows.

- **Lack of statement or keyword** (Silence), refers to clause or keyword present in the template, and not used by the participant.
- **Understatement** (Silence), refers to statements/behaviors specified following the structure presented in the template, but with less information than necessary.
- **Misspecification** (Noise), refers to statements/behaviors that have been misspecified according to the structure defined in the ontology.
- **Wrong information** (Contradiction), refers to statements which states a correct template structure, but presents wrong (or partially wrong) information for that statement.
- **Minor writing complement** (Silence), refers to the need of minor complements (or modifications) in the phrase in order to comply with the template structure or clarify the behavior's meaning.
- **High-level of abstraction** (Wishful Thinking), refers to behaviors specified in such a high-level of abstraction which not allow to assume the actual expected interaction on the UI.
- **Epic behavior** (Overspecification), refers to behaviors that encompass a wide number of implicit interactions. This kind of behavior should typically be broken into several low-level interactive behaviors. This concept is based on the concept of epics that has been introduced by Cohn (Cohn, 2004) and which refers to a large User Story that cannot be delivered as defined within a single iteration or is large enough that it can be split into smaller User Stories.

Below we present 4 tables (Table 18, Table 19, Table 20 and Table 21) detailing, for each participant, each behavior specified by him/her, the adherence of each behavior in the scale presented in the methodology, and a section of comments, where we classify the type of adherence problem identified (if any) and strive the reasons for such a kind of problem. Additionally, we propose possible corrections for problems identified in the template (we called it understandability in User Story specification), and when interactive behaviors are concerned, the correction demanded to meet the actual behavior in the ontology (we called it adherence to the ontology in User Story specification).

User Story Specification - Participant P1:

Behaviors Specified by the Participant	Possible Correction	Adherence	Comments
-	Title: Booking flights for guests	oooooo	Lack of statement or keyword. Participant did not title the story.
Narrative:	-	•••••	Participant correctly used the keyword.
As a guest	As a travel manager	••••oo	Wrong information. Participant correctly identified a role, but mistakenly specified the guest as the role who would benefit from the story, when actually it would be the travel manager.
I want airline tickets with defined time and flights	I want to search airline tickets with defined time and flights	•••••o	Understatement. Participant only forgot the action he/she expects from the system (lack of a verb)
So that I can book tickets	So that I can book tickets for guests	•••••o	Minor writing complement. Participant did not complement the benefit specifying for whom tickets will be booked.
Scenario: Searching demanded tickets	-	•••••	Participant correctly used the keyword with a name for the scenario.
Behaviors Specified by the Participant	Behaviors Defined in the Ontology	Adherence	Comments
Given I go to the site SIMBAB/TRAVEL	Given I go to “SIMBAB/TRAVEL”	•••••o	Minor writing complement. Participant added the term “the site” in the behavior “I go to” which is not present in the ontology.
When I choose the demanded flight (destination and times, TOULOUSE/PARIS, departure 7 a.m., return 7 p.m. at the same day)	When I choose “Toulouse” in the field “Departure”	••oooo	Epic behavior. Participant did not break the actions in multiple steps, having informed all the required data for searching in brackets. The behavior “I choose” is nonetheless adherent to the ontology.
	And I choose “Paris” in the field “Destination”		
	And I choose “same day” in the field “Departure Date”		
	And I choose “same day” in the field “Return Date”		
	And I choose “7 a.m.” in the field “Departure Time”		
	And I choose “7 p.m.” in the field “Return Time”		
And I choose type of traveler “Guest”	And I choose “Guest” in the field “Type of Traveler”	•••••o	Misspecification. Participant did not inform “Type of Traveler” as a field name.
And I search	And I click on “Search”	••••oo	Understatement. Participant omitted the type of behavior that will trigger the searching.

Then several propositions	Then will be displayed “List of Flights”	●●○○○	High-level of abstraction. Participant omitted the expected system behavior, only informing that the result will be “several propositions” of flights. “Several propositions” indeed will be proposed in a “List of Flights”, so this is the expected system output behavior.
And I choose the desired flights	And I choose “the desired flights”	●●●●○	Misspecification. Regardless being possible to specify a sequential input behavior in a “Then” clause, the participant is actually describing a second scenario, where he/she provides passenger’s data to effectively book the flight (an input behavior). Considering this step as part of a second scenario, the behavior of choosing the desired flight is highly adherent to the ontology.
When I inform the data concerning the traveler (name, given name, birthdate, phone, mail), and eventually the loyalty card (Flying Blue and Season Ticket)	When I inform “name” in the field “Passenger’s Name” When I inform “given name” in the field “Passenger’s Given Name” When I inform “birthdate” in the field “Passenger’s Birthdate” When I inform “phone” in the field “Passenger’s Phone” When I inform “mail” in the field “Passenger’s Mail” When I inform “loyalty card” in the field “Passenger’s Loyalty Card”	●●○○○	Epic behavior. Once more, the participant did not break the actions in multiple steps, having informed all the required data in brackets. The behavior “I inform” is nonetheless adherent to the ontology.
Then ticket waiting for validation	Then will be displayed “Ticket waiting for validation”	●●●○○	Misspecification. Once more, the participant omitted the expected system behavior, only informing that the expected result will be “ticket waiting for validation”, without describing which system’s behavior would be responsible for doing this action. Considering this is meant to be a system behavior, we should note that a behavior defining a status verification for tickets is typically a domain-specific behavior, i.e. it only refers to (and

			would make sense for) booking systems, so it is not covered by the ontology.
--	--	--	--

Table 18. User Story Specification – Participant P1.**User Story Specification – Participant P2:**

Behaviors Specified by the Participant	Possible Correction	Adherence	Comments
-	Title: Multi-destination flight search	oooooo	Lack of statement or keyword. Participant did not title the story.
-	Narrative:	oooooo	Lack of statement or keyword. Participant did not use the keyword for describing the story.
As a frequent traveler	-	••••••	Participant correctly identified the role.
I want to search for tickets, providing locations and dates for a multi-destination trip	-	••••••	Participant correctly defined a clear feature.
So that I can obtain information about rates and flight times	-	••••••	Participant correctly defined a clear business benefit.
Scenario: Multi-destination searching	-	••••••	Participant correctly used the keyword with a name for the scenario.
Behaviors Specified by the Participant	Behaviors Defined in the Ontology	Adherence	Comments
Given I go to the page “Searching Flights”	-	••••••	Participant correctly used the behavior “I go to the page”.
When I choose “Multi-destinations”	-	••••••	Participant correctly used the behavior “I choose”.
And I type “Paris” and choose “Paris, Charles de Gaulle” in the field “Departure”	-	••••••	Participant correctly used the behavior “I type and choose in the field”.
And I type “Rio de Janeiro” in the field “Destination”	-	••••••	Participant correctly used the behavior “I type in the field”.
And I choose “15/02/17” in the field “Departure Date”	-	••••••	Participant correctly used the behavior “I choose in the field”.
And I choose “20/02/17” in the field “Return Date”	-	••••••	Participant correctly used the behavior “I choose in the field”.
And I type “Rio de Janeiro” in the field “Departure”	-	••••••	Participant correctly used the behavior “I type in the field”.

And I type “Porto Alegre” in the field “Destination”	-	●●●●●	Participant correctly used the behavior “I type in the field”.
And I choose “17/02/17” in the field “Departure Date”	-	●●●●●	Participant correctly used the behavior “I choose in the field”.
And I choose “19/02/17” in the field “Return Date”	-	●●●●●	Participant correctly used the behavior “I choose in the field”.
And I click on “Search”	-	●●●●●	Participant correctly used the behavior “I click on”.
Then will be displayed the list of available flights	Then will be displayed “the list of available flights”	●●●●○	Minor writing complement. Participant only forgot quotation marks to indicate (as a variable) that “the list of available flights” is the output expected from the system.

Table 19. User Story Specification – Participant P2.**User Story Specification – Participant P3:**

Behaviors Specified by the Participant	Possible Correction	Adherence	Comments
-	Title: Checking Travel Authorizations	○○○○○	Lack of statement or keyword. Participant did not title the story.
-	Narrative:	○○○○○	Lack of statement or keyword. Participant did not use the keyword for describing the story.
As a travel manager	-	●●●●●	Participant correctly identified the role.
I want to check travel authorizations	-	●●●●●	Participant correctly defined a clear feature.
So that I can ensure the confirmed bookings	-	●●●●●	Participant correctly defined a clear business benefit.
Scenario: Listing travel authorizations	-	●●●●●	Participant correctly used the keyword with a name for the scenario.
Behaviors Specified by the Participant	Behaviors Defined in the Ontology	Adherence	Comments
Given I go to the tab “Travel Authorization”	Given I go to “Travel Authorization”	●●●●●○	Minor writing complement. Participant added the term “the tab” in the behavior “I go to” which is not present in the ontology.
When I type the “Booking Reference”	When I type “XXX” in the field “Booking Reference”	●●●●○	Understatement. Participant has omitted either the field name or the value that will be affected by (or affect) this behavior.

	OR When I type “Booking Reference” in the field “Booking Reference Field”		
And I check if the request is well registered	Then will be displayed “Request well registered”	●●○○○	Misspecification. The participant did not identify that the information about the booking registration will be provided by the system as an output. For that, a “Then” clause should be used instead of a “When”. Besides that, he/she also specified a domain-dependent behavior, without identify how the checking is supposed to be made. He/she could instead use a common interactive behavior presented in the ontology such as “will be displayed”.
Then at this time, I can know for sure (or not) the request has been taken into account	-	○○○○○	Misspecification. This is not an interactive behavior, but rather a cognitive task. This could also be considered as a business benefit of this story, and as such, it has been correctly specified in the clause “So that” in the beginning of the story.
And it's shown a tab: authorized / non-authorized	And will be displayed “Authorized” OR And will be displayed “Non-Authorized”	●●●○○	Misspecification. This step brings the expected output of the system. The participant expects to see a tab with a message signalizing whether the booking is authorized or not. This behavior has been put in a “Then” clause, indicating the participant actually understood that showing some information after his/her interaction is a system’s output. However, the participant did not realize that he/she is supposed to inform a valid * or* an invalid state, i.e. he/she should have specified a scenario in which the system would present an authorized booking, and another (if he/she wants) specifying a scenario in which the system would present an unauthorized booking.

Table 20. User Story Specification - Participant P3.**User Story Specification - Participant P4:**

Behaviors Specified by the Participant	Possible Correction	Adherence	Comments
-	Title: Searching flights to Paris	oooooo	Lack of statement or keyword. Participant did not title the story.
-	Narrative:	oooooo	Lack of statement or keyword. Participant did not use the keyword for describing the story.
As an intern	-	•••••	Participant correctly identified the role.
I want to book a flight to Paris departing on May 2nd until May 10th	-	••••oo	Wrong information. Participant mixed a feature description with data for specifying a testable scenario.
So that I can attend a seminar	-	•••••	Participant correctly defined a clear business benefit.
Scenario:	Scenario: Searching demanded tickets	•••ooo	Lack of statement or keyword. Participant did not title the scenario but used the appropriate keyword.
Behaviors Specified by the Participant	Behaviors Defined in the Ontology		
Given I'm going to book my flight	-	•ooooo	Understatement. Participant did not identify how or where the activity of booking will be performed in the system. This step is described more as an intent than as an actual behavior.
When I provide all the information	When I inform “...”	•ooooo	High-level of abstraction. Participant did not describe which kind of information should be provided for the scenario. The supposed data to be used here was mistakenly put when specifying the feature in the narrative.
And I choose search by fares	And I click on “Search by fares” OR And I choose “Search by fares” And I click on “Search”	•••••o	Misspecification. Supposing the system provides different buttons for different types of search, the participant could simply have used the behavior “click on” (supported by buttons) instead of the behavior “choose”. Otherwise, “Choose by fares” is a domain-dependent behavior, so for specifying a

			domain-independent behavior, the participant should rather have informed which option he/she would choose (or select) to “search by fares” and then submitting the search by clicking on the respective button, for instance.
Then all the available flights for the date are classified by ascending order of fares	Then will be displayed “List of available flights”	●○○○○	Misspecification. Again, the participant did lean on a domain-dependent behavior. To specify an action for verifying the arrangement of a list, it would be necessary an ontological behavior allowing to classify datasets in ascending (or even descending) order.

Table 21. User Story Specification – Participant P4.

7.6.3. Discussion

We present below a set of tables and charts consolidating different views of data extracted from the tables above. Table 22 and Figure 90 illustrate the understandability of each statement in the User Story specification. Therein, we isolated each one of the statements presented in the template and analyzed, for each participant, the dispersion of results in each degree of adherence stated in the methodology. Such a dispersion has been calculated as a median of the adherence for each stratum proposed in the experimental design.

	P1	P2	P3	P4
Title	0,00	0,00	0,00	0,00
Narrative	6,00	0,00	0,00	0,00
As a	3,00	6,00	6,00	6,00
I want	5,00	6,00	6,00	3,00
So that	5,00	6,00	6,00	6,00
Scenario	6,00	6,00	6,00	3,00
Given	5,00	6,00	5,00	1,00
When	3,00	6,00	3,50	3,00
Then	3,00	5,00	1,50	1,00

Table 22. Understandability of Each Statement in the User Story Specification.

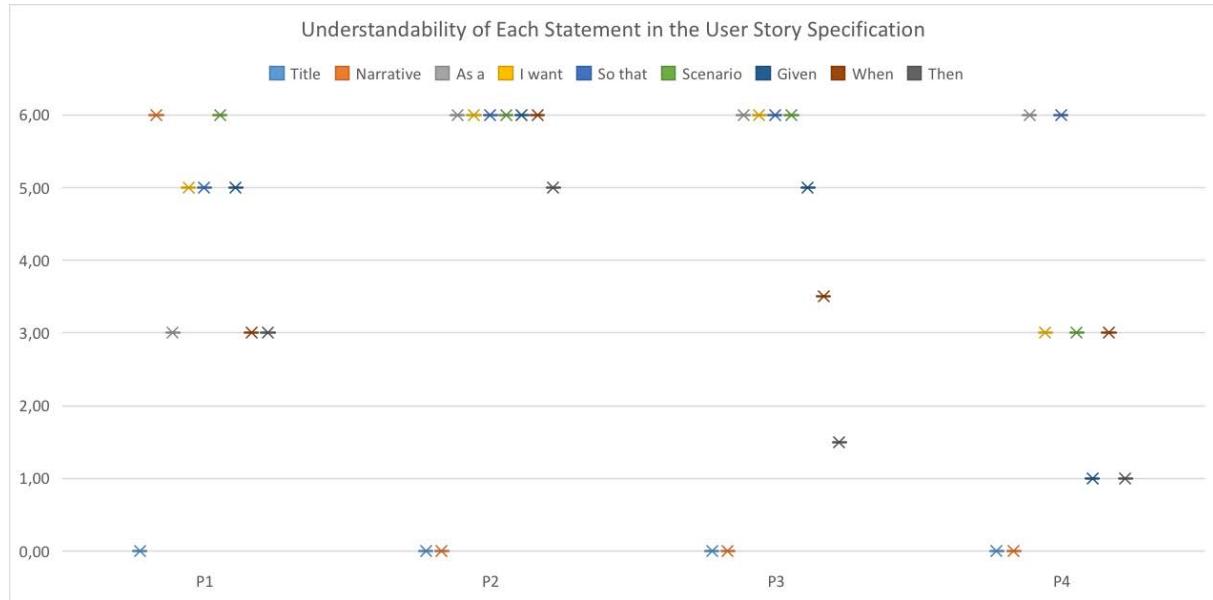


Figure 90. Understandability of Each Statement in the User Story Specification.

Table 23 and Figure 92, and Table 24 and Figure 93 illustrate the general understandability of each participant for User Story specification. The charts were built taken into account, for each participant, the number of events in each stratum ranging from a null understanding of statements to a full understanding of them. In this first chart (Figure 92) we consolidate only statements presented in the template, but not covered by the ontology as an interactive behavior (narrative section). Figure 93, on the other hand, illustrates the same information, but now considering only the adherence to interactive behaviors addressed in the ontology (scenario section). Finally, Figure 91 gives us the general understandability of User Stories based on the data from all the four participants. The chart summarizes the total amount of occurrences in each stratum of the adherence scale.

Participants	Null	Very Low	Low	Medium	High	Very High	Full
P1	1	0	0	1	0	2	2
P2	2	0	0	0	0	0	4
P3	2	0	0	0	0	0	4
P4	2	0	0	2	0	0	2

Table 23. Understandability in User Story Specification - Narrative (Number of occurrences in each stratum).

Participants	Null	Very Low	Low	Medium	High	Very High	Full
P1	0	0	3	1	1	3	0
P2	0	0	0	0	0	1	11
P3	1	0	1	1	0	2	0
P4	0	3	0	0	0	1	0

Table 24. Adherence to the Ontology in User Story Specification - Scenario (Number of occurrences in each stratum).

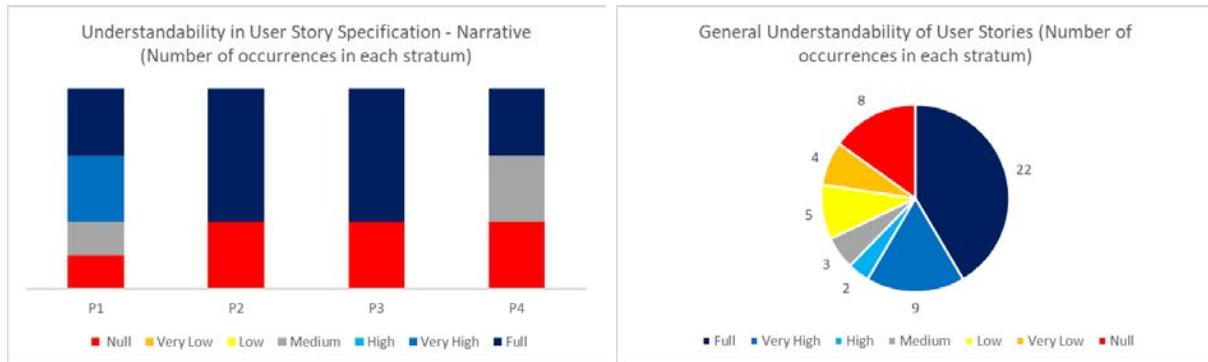


Figure 92. Understandability in User Story Specification - Narrative (Number of occurrences in each stratum).

Figure 91. General Understandability of User Stories (Number of occurrences in each stratum).

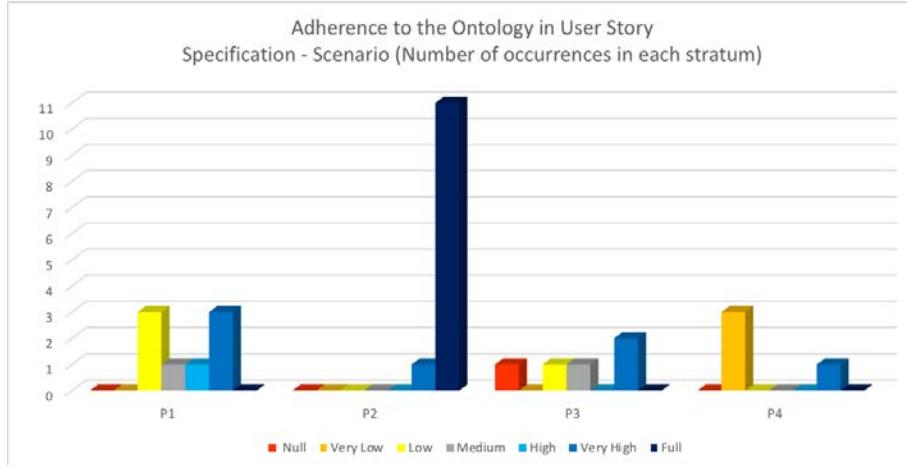


Figure 93. Adherence to the Ontology in User Story Specification - Scenario (Number of occurrences in each stratum).

RQ 1. Are participants able to read/understand a basic User Story template and use it to write their own stories?

First of all, concerning the understandability of User Stories (narrative section), we notice the majority of participants neglected in titling and using the keyword “Narrative” in the beginning of the stories. Only P1 used the keyword, but even him/her did not title the story. We are not sure

about the main reasons for that. In a first approach, it seems more like a lack of attention from the participants. All participants, except P1, identified a correct role (statement “As a”) for the stories. P1 correctly identified a role, but mistakenly specified the guest as the role who would benefit from the story, when actually it would be the travel manager. Concerning the feature description (statement “I want”), we noticed a very good understanding of this statement, with participants ranging from 5 to 6 in our scale, except P4. P4 has mixed the feature description with data for specifying a testable scenario. Concerning the business benefit expected from the feature (statement “So that”), all participants shared a very good understanding as well, ranging likewise from 5 to 6.

We have also observed in these charts that the stories produced by P2 and P3 had identical results, both with a majority of full adherence and a medium-to-low number of null adherence statements. These last ones due to the lack of “Title” and “Narrative” sections of the story. P1 had a low level of null (absence of title) and medium (wrong information when identifying the role) adherent statements, and a clear majority of very-high and full adherent statements. P4 had an equal mix of null, medium and full adherent statements, with problems varying from absence of keywords or sections until the presence of wrong information.

RQ 2. Which is the vocabulary the participants make use when writing their own User Stories?

Concerning the adherence to the ontology in User Stories specification, i.e. the adherence in the section “Scenarios:”, we have noticed all the participants titled their scenarios, except P4, that regardless use the right keyword, not titled his/her scenario. For the statement “Given”, we have observed a tendency in users specifying more information to define where they are going to access some feature. The ontology has specified a generic behavior named “I go to” and a variation to “I go to the page”. However, while P1 used this convention, P2 and P3 have specified respectively, “I go to the site” and “I go to the tab”, so at this point, somehow the ontology could be enriched to recognize those variants as well. P4, on the other hand, specified a very generic behavior (“Given I’m going to book my flight”), not identifying how or where the activity of booking will be performed in the system. This step is described more as an intent than as an actual behavior. For this statement then, P1, P2 and P3 scored a very high adherence (between 5 and 6), while P4 scored a very low adherence (1).

Concerning the statement “When”, we notice a mid-range understanding (between 3 and 3,50) for P1, P3 and P4, and a full understanding (6) for P2. P1 produced what we classify as epic behaviors, providing in a same step, several independent actions to be performed on the UI. P3 shared a well formulated step with a misspecification. The participant either confused an output information (that should be specified in a “Then” statement) or specified a domain-dependent behavior, not supported by the ontology scope. P4 specified a behavior with a high-level of abstraction without describing which kind of information should be provided, along with a domain-dependent behavior.

Finally, concerning the statement “Then”, we notice a pretty low understanding (between 1 and 1,50) for P3 and P4, a mid-range understanding (3) for P1 and a very high understanding (5) for P2. P3 specified a kind of cognitive task in his/her first “Then” statement, defining much more a business benefit than an interactive task. His/her second “Then” statement brings a misspecification that despite specifying the expected output of the system, it does not comply with a single valid or invalid state, expressing both states in the same expected output. P4 wrote a single misspecified “Then” statement, indicating once more the use of a domain-dependent behavior. P1 specified “Then” statements with a high-level of abstraction, along with small misspecifications, being one of them related to the use of a new input interaction, and another related to the use of

a domain-specific behavior. P2 committed a really minor writing mistake, only forgetting to use quotation marks to indicate a variable in the interactive behavior.

We have also observed P2 wrote a very high-adherent story with only some minor deviations, especially when describing the narrative. In contrast, P4 wrote a very low-adherent story with the majority of statements classified as having a very low adherence. P1 had half of low and medium adherent statements along with half of high and very high ones. P3 had a slight majority of statements flirting with the low-level stratum (a mix of null, low and medium) and the remaining ones classified as very high adherence.

Looking at the general understandability of User Stories, we notice however a large majority of statements classified as full or very high adherence to the template. From a total of 53 statements, 33 (62,26%) were classified in the top stratum (full, very high, and high adherence), 3 (5,66%) in the medium stratum, and the remaining 17 (32,08%) in the bottom stratum (low, very low, and null adherence).

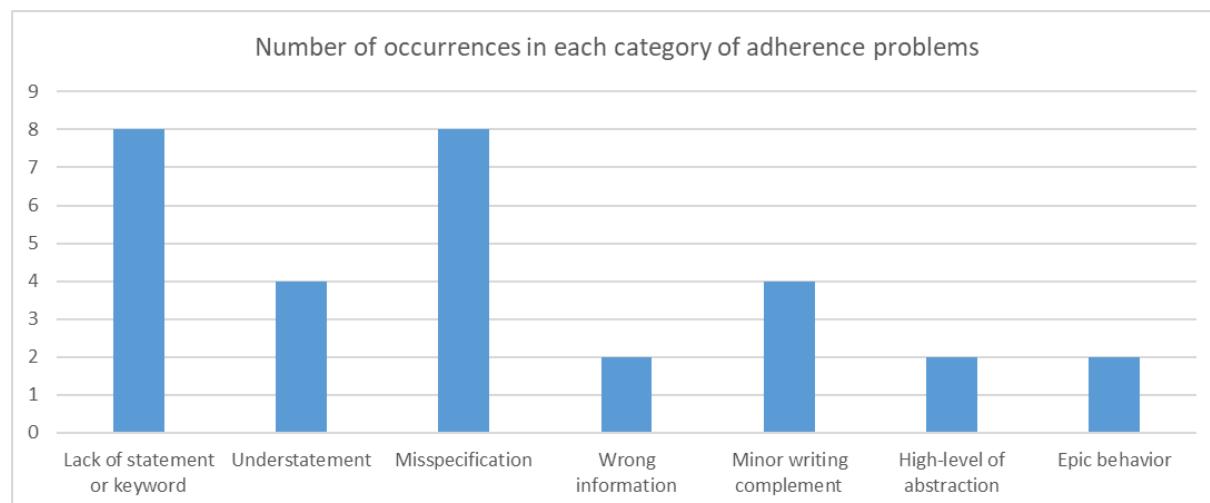


Figure 94. Number of occurrences in each category of adherence problems.

We have also analyzed the type of adherence problems found in the stories specified by the participants. As explained in the adherence analysis section above, we have identified 7 types of problems as follows: lack of statement or keyword, understatement, misspecification, wrong information, minor writing complement, high-level of abstraction, and epic behavior. Figure 94 brings the number of occurrences in each category.

In a total of 30 adherence problems identified, we can observe in the chart that the most common types of adherence problems have been the “lack of statement or keyword” and the “misspecification” with 8 occurrences each. It is more than 50% of the problems found (53,33%). “Wrong information”, “high-level of abstraction”, and “epic behavior” were, on the other hand, the types of adherence problems less observed in the participants’ User Stories. With a total of 2 occurrences each, they represent singly no more than 7% of occurrences (6,67%). “Understatement” and “Minor writing complement” complete the set, with each type reporting 4 occurrences, i.e. 13,33% of occurrences each.

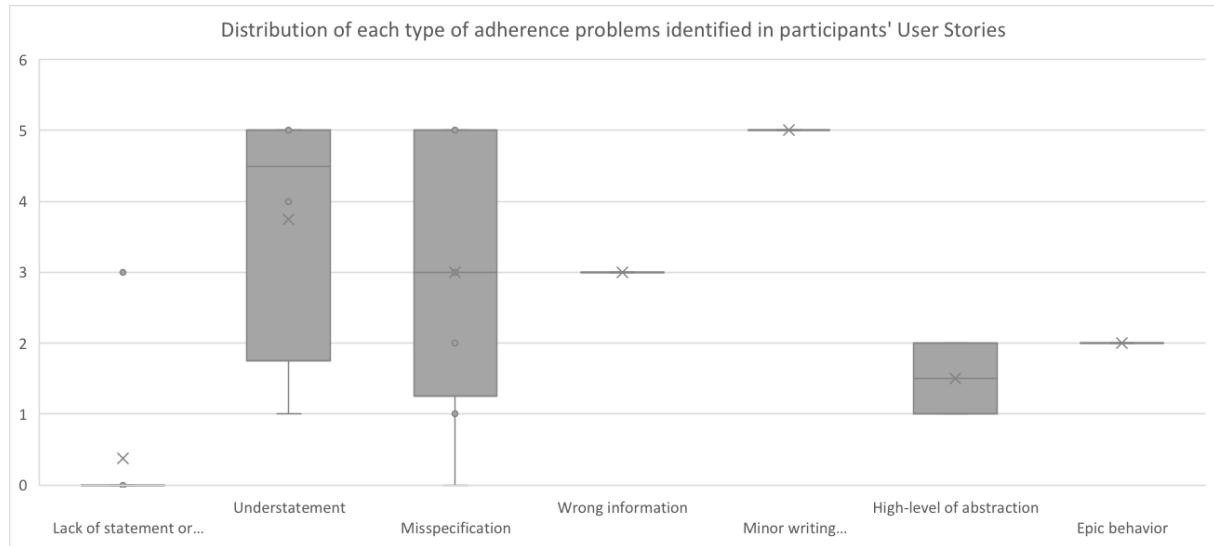


Figure 95. Boxplot of each type of adherence problems identified in participants' User Stories.

Figure 95 brings the boxplot of each type of adherence problems. Y-axis brings the scale of adherence defined for this study and presented in the methodology section. We observe therein that the category “misspecification” had the largest dispersion, ranging from 0 (null, with the lower quartile near 1) to 5 (very high, coinciding with the upper quartile), with a median (and a mean) at the medium stratum of adherence problems. “Understatement” had the second largest dispersion, ranging from 1 (low, with the lower quartile near 2) to 5 (very high, coinciding with the upper quartile) with a median at the top stratum (4,5) and the mean near the level 4 of adherence. “High-level of abstraction” comes next with a dispersion between 1 (very low, coinciding with the lower quartile) to 2 (low, coinciding with the upper quartile), with median and mean in 1,5. “Lack of statement or keyword”, “wrong information”, “minor writing complement”, and “epic behavior” had no dispersion, and achieved respectively a median of 0, 3, 5 and 2. Equal results have been observed for the mean of these types of adherence problems (“lack of statement or keyword” had a mean slightly above the median). Outliers have been observed for “lack of statement or keyword” with just an occurrence of an adherence problem classified in the medium stratum with all the others classified in the null stratum.

7.7. Findings and Implications

Based on the results discussed above, we can highlight some important findings about the writing profile of User Stories specified by the participants. The wide dispersion of adherence problems classified as “misspecification” means participants had a varied level of compliance for the problems found in this class, since a slight mistaken identification of fieldnames until heavy domain-dependent behaviors. By the way, we clearly noticed, by analyzing the type of adherence problems committed by the participants, that the specification of domain-dependent behaviors was one of the most frequent issues. Even with the high number of misspecifications, participants, most of time, completely understood the purpose of a scenario, but as they did not know there was a set of predefined interactive behaviors supposed to be followed, they freely specified the desired behavior describing exactly what they expected from the system. This fact is confirmed by the medium to low adherence in the “When” and “Then” statements of the story, where typically reside the most interactive behaviors in a scenario, and consequently, where the ontology is more used to specify them.

Concerning the general understandability of User Stories, we notice a clear concentration of occurrences in the top stratum (62,26%), which signalizes an overall very good understanding of User Stories in the proposed template and a limited but spontaneously use of our predefined interactive behaviors presented in the ontology. Analyzing each statement of the stories individually, we also notice a clear concentration of occurrences in the top stratum, exception made for the aforementioned “When” and “Then” statements which are dispersed mostly between the medium and the low stratum, and for “Title” and “Narrative” statements that were almost always omitted by the participants, which occasioned consequently a null adherence for both of them with only one exception.

“Understatement” problems, despite their high dispersion, presented a median at the top stratum (4,5), which means the level of noncompliance for this kind of problem is very low, so we conclude participants made, in general, just slight deviations from the proposed template. “Lack of statement or keyword”, despite the high number of occurrences, was primarily found in the “Title” and “Narrative” statements that were frequently omitted by the participants, which explains the prevalence of null adherence for this type of problem. “High-level of abstraction” and “epic behaviors” presented problems with a low level of adherence to the proposed template once these kinds of problems are associated to descriptions with a low level of interaction details, which is opposed to what is defined in the ontology. As expected, “minor writing complement” had a very high rate of adherence with behaviors presenting only minor deviations from the proposed template.

By looking for individual causes of the problems found, we observed P1 and P3 signalized a medium understanding of the structure and the purpose of acceptance scenarios in the stories. P1 and P3 stories performed primarily at the top stratum (very high and full adherences for P1, and full adherence for P3) for the narrative section. For the scenario section however, P1 mixed a performance of half occurrences at the top stratum (high and very high adherences) and half at the medium to low stratum (medium and low adherences), while P3’s story performed primarily at the medium to low stratum (null, low and medium adherences for 2/3 of occurrences) with the remaining occurrences being classified as very high. P1 and P3 had the largest number of behaviors marked as “misspecification”, confirming a particular difficulty to assimilate some structures of User Stories in the proposed template, mixing primarily the writing of some domain-dependent behaviors with “understatement” and “minor writing complement”.

Epic behaviors have been only specified by P1. In the context it has been made, a sequence of data input in a form, this error could signalize the need of tables to enter a set of data in forms. This kind of solution has been proposed by the FIT Framework⁶, however it is not covered by our ontology so far. “High-level of abstraction” however has been observed in stories written by P1 and P4. P4’s low performance (2/3 of occurrences classified in the null and medium adherences for the narrative section, and the clear majority of occurrences classified as very low for the scenario section) could find an explanation in the participant’s lack of experience in the business processes at our institute (just a month), despite having 4 years of experience working for other companies.

Analyzing the greatest performance and the highest adherence of P2’s stories (primarily at the top stratum – very high and full adherences for the narrative section – with an overwhelming majority of full adherent statements – with a single very high adherence – for the scenario section), and being the second younger participant besides the second more experienced one, we wonder about the role played by the sum of age and experience factors in the willingness and commitment

⁶ <http://fit.c2.com/>

to adopt new ways of work. P2 had clearly the better performance with the lowest ratio age/experience in the group.

These findings bring us some opportunities for improving our current set of interactive behaviors in the ontology. As stated before, the adoption of tables with data examples together with the ontology could reduce the workload of describing input of data in forms and stimulate a complete specification by users. The ontology could also be enriched to recognize variants for the same interactive behavior. Participants of this study specified some behaviors very close to the ontology statements, but with minor variants. The ontology has indeed a restricted vocabulary. Even mapping synonyms for some specific behaviors, it does not provide any kind of semantic interpretation, i.e. behaviors must be specified on stories exactly as they were defined in the ontology. Further studies on Natural Language Processing (NLP) techniques might help to improve the process of specification adding more flexibility to write scenarios that could be semantically interpreted to meet the behaviors described in the ontology. This issue is certainly a worthwhile topic for further research.

Another aspect to be considered is the high number of domain-dependent behaviors specified by the participants. This point us out the need of considering a still higher level of descriptions for our behaviors. Domain-specific behaviors have the disadvantage of being dependent on the jargon used for each type of business processes, which would implicate in developing different ontologies for different business processes, with each one encompassing the proper jargon of each domain. Domain-specific ontologies nonetheless could act as a top layer in a multi-layer ontology architecture to allow the use of multiple domain ontologies associated to the current domain-independent ontology, which would remain describing only the fundamental interactive behaviors for a given environment.

Going back to our stated research questions, we can conclude that results point to a high level of understandability of User Stories when their structure is considered (narrative section and scenario/given/when/then intents), i.e. the participants were able to read/understand a basic User Story template and use it to write their own stories (RQ 1). We can also conclude the vocabulary of the interactive behaviors described in the ontology was spontaneously used, even without a specific prior training in the adopted vocabulary (RQ 2). As we highlighted in chapter 4, the vocabulary chosen to describe such interactive behaviors emerged from our previous experiences in scenarios specified for real projects, so we can infer it reflects somehow a natural writing vocabulary for stakeholders. Nevertheless, this vocabulary could eventually be extended in the future to support more representative phrases or expressions. Finally, we identified a total of 7 types of adherence problems in the participants' User Stories, being "lack of statement or keyword" and "misspecification" the most common ones.

7.7.1. Threats to Validity

Generalization of results. We have selected a representative group of participants as Product Owners (POs) in a system for booking airline tickets for business trips. Such kind of system has usually a strong search-based feature, once they are centered in providing and comparing rates, times and availability of flights given a set of provided parameters. However, as the ontology in which we based our analyses is designed for domain-independent interactive behaviors, we assume our results would be reproduced in other interactive systems domains. The profile and previous experiences of the participants could, nonetheless, bring different results. Experiments involving Product Owners previously introduced to User Stories and/or test automation could bring different and less frequent adherence problems.

Length of the sample. We have conducted this experiment with 4 participants that could eventually assume a role of Product Owners in a typical scenario of software development. Our results are certainly limited to the profile and experience of these four participants. Experiments conducted with a bigger sample could bring different adherence problems and/or reduce the variability of occurrences when looking to the whole group. It could eventually bring more homogeneous results.

Absence of training. This experiment has been conducted without training the participants in the adoption of interactive behaviors presented in the ontology. As stated before, this decision was made because one of the goals of this study was to investigate in which extent the interactive behaviors described in the ontology would be perceived as useful enough to be spontaneously reproduced by the participants. Experiments involving prior training in the vocabulary used in the ontology would certainly bring different results due to the background knowledge. However, such results would not capture the spontaneous factor of users choosing their own vocabulary to express their interaction needs. This factor is useful to identify the suitability of the predefined interactive behaviors to naturally express the user's intents.

Possible interpretation bias. Both the conduction of the experiment and the interpretation and analysis of the results have been made by us. So, it is possible there has been a bias in the interpretation of such results, especially when scaling the adherence of each statement in the stories produced by the participants. At this point, the results are being cross-checked by an independent reviewer in an attempt to reduce such a bias and mitigate this threat.

7.8. Conclusion

This chapter presented a study we have conducted to evaluate the understandability of User Stories by potential Product Owners, represented by team members of the travel department in our research institute. When analyzing the adherence of the User Stories produced by the participants, the study has shown they had an overall good understanding of User Stories statements and structure, and a moderate-to-high spontaneous understanding of the implicit ontological patterns presented in the template they received.

An important remark we can notice is that all the stories written by the participants are, in general, well suited to communicate a business intent or even a concrete feature of the system, if testing automation is not a concern. Other studies (Wautelet *et al.*, 2014) have investigated the suitability of different templates for User Stories and how they could be improved to set an agreement in their semantics and methodological elements, which could help to improve communication between stakeholders. However, our focus in this study is mainly to investigate how far off such stories are from the specification of our common ontological behaviors which allow us running automated testing.

We also consider this study is highly reproducible once the ontology has general use intent and specifies domain-independent interactive behaviors. As such, similar studies could be conducted to evaluate the adherence of the ontology in different contexts. We also consider our results can be generalized given the need of describing low level interactive behaviors for automated User Stories would permeate software testing activities in any domains.

In short, we can summarize our findings as follows:

- Concerning the general understandability of User Stories, we notice a clear concentration of occurrences in the top stratum (62,26%), which signalizes an overall

very good understanding of User Stories by the participants, with a limited but spontaneous use of our predefined interactive behaviors presented in the ontology.

- We clearly noticed, by analyzing the type of adherence problems committed by the participants, that the specification of domain-dependent behaviors was one of the most frequent issues.
- “Understatement” problems, despite their high dispersion, presented a median at the top stratum (80% of adherence), which means the level of noncompliance for this kind of problem is very low.
- “Lack of statement or keyword”, despite the high number of occurrences, was primarily found in the “Title” and “Narrative” statements that were frequently omitted by the participants.
- “High-level of abstraction” and “epic behaviors” presented problems with a low level of adherence to the proposed template.

As future works, we wonder about the impact of absence of training on the results. New studies should be conducted to evaluate the potential impact of prior training sections with the participants concerning the predefined interactive behaviors presented in the ontology before conducting the experiment. Regardless this current study has as objective to evaluate the spontaneous use of such behaviors by the participants, our hypothesis is that prior training could probably enhance the level of adherence of the stories produced. We also wonder whether results could have been influenced by the high-level of experience of the participants in the business process. Studies with a larger sample and/or with participants with experience in User Stories instead could attenuate this factor and bring different results.

Chapter 8

Case Study II - Assessing User Interface Design Artifacts

Summary

In this chapter, we reuse the User Stories created by our potential Product Owners in chapter 7. This second study proposes to redesign (by the means of a reverse engineering of the current software system) task models and user interface prototypes to assess their compatibility with the user requirements expressed in such a system. To do that, we apply our proposed testing approach to check the consistency of such artifacts along with the final user interface of the current software system. The aim of this study is to identify which kind of inconsistency problems we can find with our testing approach and to demonstrate its potential.

The first section of this chapter (section 8.1) presents the case study design, detailing how the study was planned and executed. The second section (section 8.2) presents the set of complementary User Stories we have developed to support the design and testing of the artifacts developed for the case study. The third section (section 8.3) adds a group of selected test cases with the aim of helping to validate such stories. The following sections present the modeling and testing results for each one of the assessed artifacts: task models (in section 8.4), Balsamiq prototypes (in section 8.5), and final UIs (in section 8.6).

In the section 8.7, we build a traceability mapping to follow the inconsistencies found in each one of the target artifacts. Such mapping shows an edge-to-edge overall view of the testing scenarios, signalizing where a given step has failed in each artifact and why. We finish by presenting our findings and lessons learned in the section 8.8, as well as our conclusions on the effectiveness of our testing approach and the impact of the inconsistencies identified in the assessment of artifacts (section 8.9).

The present chapter considers the outputs of the study presented in the previous chapter by exploiting new User Stories and modeling new reengineered user interface design artifacts for testing. The following sections present how we have designed such a study, and in which extent the results helped us to analyze the kind of inconsistency problems we can identify with the testing approach we propose in this thesis.

8.1. Case Study Design

To conduct this study, we have refined the User Stories written by the participants to simulate the assessment of user interface design artifacts obtained by reengineering the current system for booking business trips presented in chapter 7. To do that, we have studied the current implementation of user requirements in this current system, and by applying *reverse engineering* (Chikofsky and Cross II, 1990), we redesigned the appropriate task models and user interface prototypes for the system. The aim of this software reengineering is to have such artifacts to run our tests and verify in which extent our approach is able to identify inconsistencies between them.

Our motivation for this study is to understand which kind of inconsistencies we can identify by using this approach. Therefore, the main objectives of this case study are:

- To demonstrate the potential of the approach to assess user interface design artifacts;
- To identify which kind of inconsistencies we are able to point out by running our testing approach in the set of reengineered artifacts for the business trip case study;
- To exemplify our approach as presented in chapters 5 and 6.

To achieve these goals, we planned our study divided in 6 steps as follows:

- Step 1: Format and add new User Stories based on the outputs from the previous study and based on the current system implementation.
- Step 2: Add test cases to these User Stories.
- Step 3: Reengineer task models for the current system and run our approach to test the developed scenarios.
- Step 4: Reengineer user interface prototypes for the current system and run our approach to test the developed scenarios.
- Step 5: Run our approach to test the final user interface of the current system with the same developed scenarios.
- Step 6: Trace the results and verify the extent of inconsistencies we were able to identify in these multiple artifacts.

All these steps were performed by ourselves after conducting the previous study with the POs in the business travel department. With the aim of simulating a software development lifecycle, we firstly developed an initial version of User Stories and their test cases to act as our user requirements and acceptance criteria. We then reengineered initial versions of the respective task models and user interface prototypes to model such requirements. After getting ready a first version of task models, we extracted a representative set of scenarios from them. By following our strategy for testing, we run this initial version of User Stories to the initial set of scenarios extracted from task models. Results were then evaluated, and we could observe the type of inconsistency we succeeded identifying. As the strategy we follow for testing scenarios in both task models and User Stories parses all the steps of each scenario at once, the first round of results is obtained with a single battery of tests.

Following this step, we run the same initial version of User Stories to initial versions of user interface prototypes designed using Balsamiq. Unlike the strategy for testing task models, the strategy we follow for testing user interface prototypes and final UIs parses each step of the scenario at a time, so if an error is found out, the test stops until the error is fixed. That requires to run several batteries of tests until having the entire scenario tested. It leads us to fix all the inconsistencies step-by-step, and consequently to get fully consistent scenarios at the end of running. However, when analyzing the reason related of each inconsistency, we can eventually conclude the origin of the inconsistency is actually in the specification of the step in the User Story scenario, and not in the artifact itself. As a result of such, to fix such an inconsistency, steps of User Story scenarios may also be modified along the battery of tests to comply with a consistent specification of the user requirements. An immediate consequence of this fact is that the steps used to test a given version of an artifact can be different than that ones used to test another artifact previously. It means that regression tests are crucial to ensure that a given modification in the set of User Stories scenarios did not break some previous test in other artifacts and made some artifact (that so far was consistent with the requirements) inconsistent again.

We applied the same strategy to test the final user interfaces, once essentially they are fully-fledged versions of previous user interface prototypes. The difference here is that, by applying a reverse engineering approach, we assume that the released version of the current booking system (and evidently its final UI) represents the unequivocal statement of the user requirements, once for the purpose of this study, we cannot modify them. As such, we have not the opportunity to eventually redesign the final UI to comply with the User Stories we developed. As a consequence, all the identified inconsistencies necessarily resulted in modifications in the steps, not in the final UI.

Finally, we analyzed the results of testing in each artifact by mapping such results to identify the trace of each inconsistency throughout the artifacts. That gave us a complete traceability overview of each step of the User Stories in the target artifacts. During the execution of each step of testing described above, we have collected and identified the reasons of failure in the mentioned artifacts in order to answer our research question concerning the kind of inconsistencies we are able to identify with this proposed approach. Such results allowed us to evaluate the effectiveness of the approach and to identify future improvement opportunities.

8.2. Formatting and Adding New User Stories

Based on the stories identified during the interview sections and presented in the previous chapter, we formatted them by following the ontology vocabulary and the template proposed in chapter 3. We also added some new stories that we have identified as user requirements in the current software system for booking business trips in our institute. In the User Story "Flight Tickets Search" (Figure 96), we have scenarios for searching flights for a roundtrip (with and without selecting all the optional fields), a one-way trip, and a multidestination trip. In the second User Story "Select a suitable flight" (Figure 97), we have scenarios for selecting suitable flights according to the results of searching. Finally, in the third User Story "Confirm Flight Selection" (Figure 98), we have scenarios for confirming or declining the respective trips.

First User Story: informing multiple criteria to search flights:

```
User Story: Flight Tickets Search

Narrative:
As a IRIT researcher
I want to be able to search air tickets for my business trips, providing
destinations and dates
So that I can obtain information about rates and times of the flights.

Scenario: Successful Roundtrip Tickets Search
Given I go to "Flight Search"
When I select "Round Trip"
And I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field
"Departure"
When I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field
"Destination"
And I set "Sam, Déc 1, 2018" in the field "Departure Date"
When I set "Lun, Déc 10, 2018" in the field "Arrival Date"
And I submit "Search"
Then will be displayed "2. Sélectionner un voyage"

Scenario: Successful Roundtrip Tickets Search With Full Options
Given I go to "Book Flights"
When I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field
"Departure"
And I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field
"Destination"
When I set "Sam, Déc 1, 2018" in the field "Departure Date"
```

```

And I set "08:00" in the field "Departure Time Frame"
When I choose "Round Trip"
And I set "Lun, Déc 10, 2018" in the field "Arrival Date"
When I set "10:00" in the field "Arrival Time Frame"
And I choose the option of value "2" in the field "Number of Passengers"
When I set "6" in the field "Timeframe"
And I select "Direct Flights Only"
When I choose the option of value "Economique" in the field "Flight Class"
And I set "Air France" in the field "Companies"
When I submit "Search"
Then will be displayed "2. Sélectionner un voyage"

Scenario: Successful One-way Tickets Search
Given I go to "Book Flights"
When I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "Departure"
And I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"
When I set "Sam, Déc 1, 2018" in the field "Departure Date"
And I choose "One-way Trip"
When I submit "Search"
Then will be displayed "2. Sélectionner un voyage"

Scenario: Successful Multidestination Tickets Search
Given I go to "Book Flights"
When I choose "Multidestination Trip"
And I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "Departure"
When I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"
And I set "Sam, Déc 1, 2018" in the field "Departure Date"
When I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Departure"
And I inform "Nice" and choose "Nice, Côte D'Azur (NCE)" in the field "Destination"
When I set "Sam, Déc 10, 2018" in the field "Departure Date"
And I submit "Search"
Then will be displayed "2. Sélectionner un voyage"

```

Figure 96. User Story “Flight Tickets Search”.

Second User Story: selecting flights from a given list of available flights:

```

User Story: Select a suitable flight

Narrative:
As a IRIT researcher
I want to get a list of compatible flights (including their rates and times) in
accordance with my search criteria
So that I can select a suitable flight based on my needs.

Scenario: Select a Return Flight Searched Without Full Options
Successful Roundtrip Tickets Search
Given "Availability Page" is displayed
When I click on "No Bag" referring to "Air France 7519"
And I click on "No Bag" referring to "Air France 7522"
When I click on "Book"
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s)
aérien(s)."

Scenario: Select a Return Flight Searched With Full Options
Successful Roundtrip Tickets Search With Full Options
Given "Availability Page" is displayed
When I click on "No Bag" referring to "Air France 7519"
And I click on "No Bag" referring to "Air France 7522"
When I click on "Book"
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s)
aérien(s)."

```

Scenario: Select a One-way Flight
 Successful One-way Tickets Search
Given "Availability Page" is displayed
When I click on "No Bag" referring to "Air France 7519"
And I click on "Book"
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."

Scenario: Select a Multidestination Flight
 Successful Multidestination Tickets Search
Given "Availability Page" is displayed
When I click on "No Bag" referring to "Air France 7519"
And I click on "No Bag" referring to "Air France 7700"
When I click on "Book"
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."

Figure 97. User Story "Select a suitable flight".

Third User Story: confirming (or declining) a selected trip:

User Story: Confirm Flight Selection

Narrative:
As a IRIT researcher
I want to get all the required data to confirm my flights
So that I can check the information, the fare rules and then finalize my booking.

Scenario: Confirm a Flight Selection
 Select a Return Flight Searched Without Full Options
Given "Confirmation Page" is displayed
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."
And I click on "Finalize the trip"
Then will be displayed "Votre voyage a été confirmé!"

Scenario: Confirm a Flight Selection (Full Version)
 Select a Return Flight Searched With Full Options
Given "Confirmation Page" is displayed
When I choose "I accept the General Terms and Conditions."
And I click on "Finalize the trip"
Then will be displayed "Votre voyage a été confirmé!"

Scenario: Confirm a Flight Selection for a One-Way Trip
 Select a One-way Flight
Given "Confirmation Page" is displayed
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."
And I click on "Finalize the trip"
Then will be displayed "Votre voyage a été confirmé!"

Scenario: Confirm a Flight Selection for a Multidestination Trip
 Select a Multidestination Flight
Given "Confirmation Page" is displayed
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."
And I click on "Finalize the trip"
Then will be displayed "Votre voyage a été confirmé!"

Scenario: Decline a Flight Selection
 Select a One-way Flight
Given "Confirmation Page" is displayed
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."
And I click on "Decline the trip"
Then will be displayed "Votre voyage a été annulé!"

Figure 98. User Story “Confirm Flight Selection”.

8.3. Adding Testing Scenarios

By analyzing the business rules for this kind of system, we added two common test cases (already explored in chapter 3) to the first User Story (“Flight Tickets Search”). These two test cases (Figure 99) scenerize two error situations when trying to book a trip: (1) try to book it more than one year in advance, and (2) try to book a return flight before the departure flight.

```
Scenario: Search for Flights More Than One Year in Advance
Given I go to "Book Flights"
When I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field
"Departure"
And I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field
"Destination"
When I set "Dim, Déc 1, 2019" in the field "Departure Date"
And I choose "One-way Trip"
When I submit "Search"
Then will be displayed "Erreur : Vous devez choisir une date de départ ultérieure
comprise entre 4 heures et 11 mois. Veuillez sélectionner une autre date. (10032)"

Scenario: Search for a Return Flight Before a Departure Flight
Given I go to "Book Flights"
When I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field
"Departure"
And I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field
"Destination"
When I set "Lun, Déc 10, 2018" in the field "Departure Date"
And I choose "Round Trip"
When I set "Sam, Déc 1, 2018" in the field "Arrival Date"
And I submit "Search"
Then will be displayed "Erreur : La date de retour ne peut pas être antérieure à la
date de départ."
```

Figure 99. Test scenarios for the User Stories.

8.4. Modeling and Assessing Task Models

Task models have been developed for this study by using the HAMSTERS tool. As we have focused in the process of searching and demanding a booking of flights (without focusing on the administrative procedure to confirm the flight), the four models below have been divided to cover the processes of searching the flights, informing a flight leg (or a new flight leg in case of a multideestination trip), and choosing and confirming (or declining) the selected trip.

Figure 100 presents the task model for searching flights using Travel Planet (the current system of booking). All the tasks have been designed to be performed by end users of the system, i.e. researchers from our institute booking their own flights, or the travel department team booking flights on behalf of the researchers. The Search Flight feature encompasses accessing the search flight page (task “Go to Book Flights”), informing at least one flight leg (abstract task “Inform a Flight Leg”), providing flight data for searching (abstract task “Provide Data to Search”), submitting the search (task “Submit Search”), and verifying the resultant list of flights (abstract task “Verify List of Flights”). These four tasks are supposed to be performed exactly in this sequence, so the operator “Enable” has been used.

To inform a flight leg (Figure 101), the user is supposed to inform departure and destination data. Such data include informing a departure and arrival cities and based on a list of available airports in those cities, selecting the ones he/she wants. Both tasks are mandatory and should be

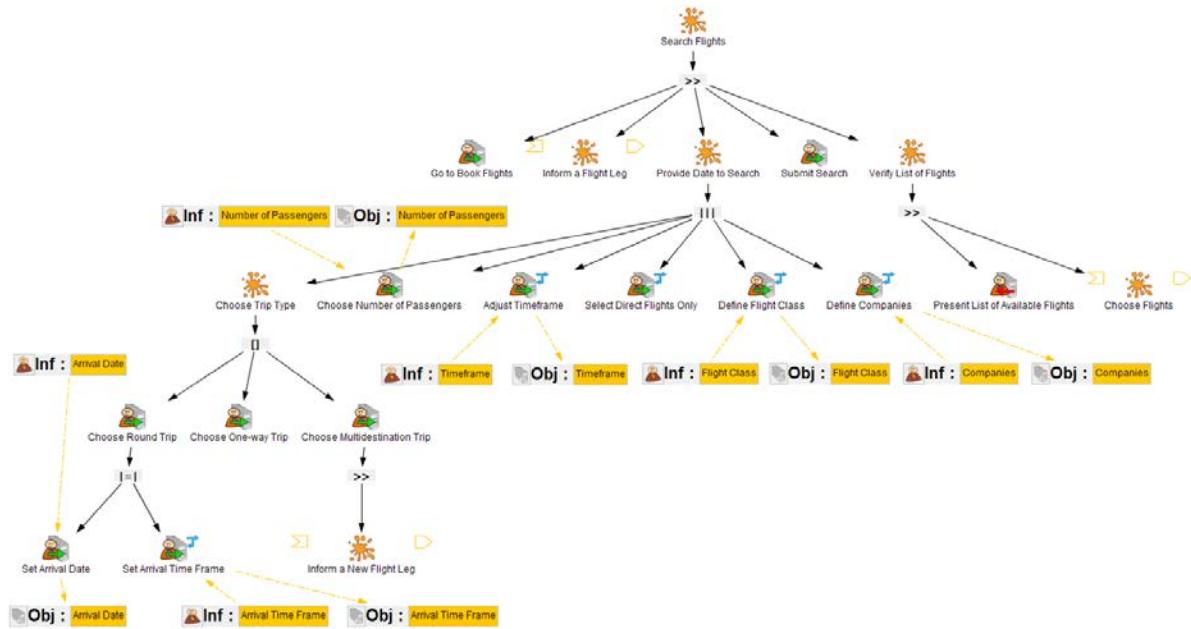


Figure 100. Task Model for Searching Flights using Travel Planet.

performed sequentially, so the operator “Enable” has been used. After selecting the airports of departure and arrival, the user must set in any order the departure date and the departure time frame, being these last one an optional task.

Going back to providing flight data to search, the user can perform in any order (operator “Order independent”) the following tasks: “Choose Trip Type”, “Adjust Timeframe”, “Select Direct Flights Only”, “Define Flight Class”, and “Define Companies”, being the four last tasks optional. For choosing trip types, the user has three options. If a round-trip is chosen, then a sequence of two order-independent tasks can be performed by the user: “Set Arrival Date” and “Set Arrival Time Frame”, being this last one optional. If a multi-destination trip is chosen, then the user must inform at least one more flight leg (abstract task “Inform a New Flight Leg”), performing the same interactive tasks from “Inform a Flight Leg”. Finally, if a one-way trip is chosen, there is no additional tasks to perform for the abstract task of choosing a trip type. For

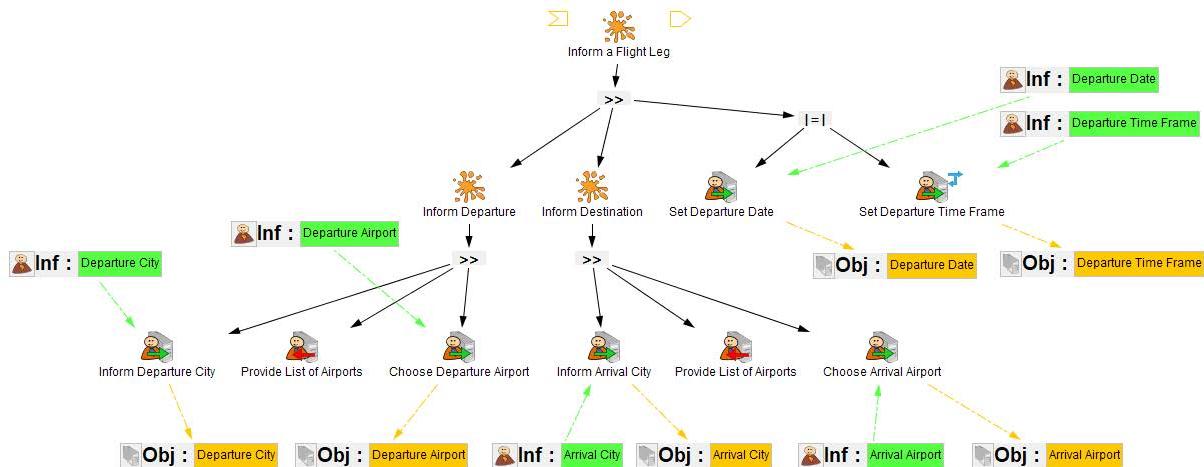


Figure 101. Task Model for Informing a Flight Leg in Travel Planet.

all the input tasks, notice that the data handling is shown with information being provided as input for the task and objects being the output of these tasks.

After providing the data, the user can submit the search and get, as a result, a list of available flights matching his/her criteria. At this point, the system returns such a list and the user can then pick one of the available flights and confirm or decline his/her booking. For performing such tasks, the abstract task “Choose Flights” has been modeled (Figure 102). To get it done, the user must firstly evaluate the availability of flights (which is a cognitive analysis task), choose the desired flight (which is a cognitive decision task), and then select the desired flight (which is indeed an interactive input task). Optionally the user can change the fare profile for the flight he/she has chosen, and then submit his/her choice. Lastly, the user checks the selected flights (a cognitive task) and verify the fare conditions (a perceptive task). He/she then finally chooses between decline the booking or conclude it.

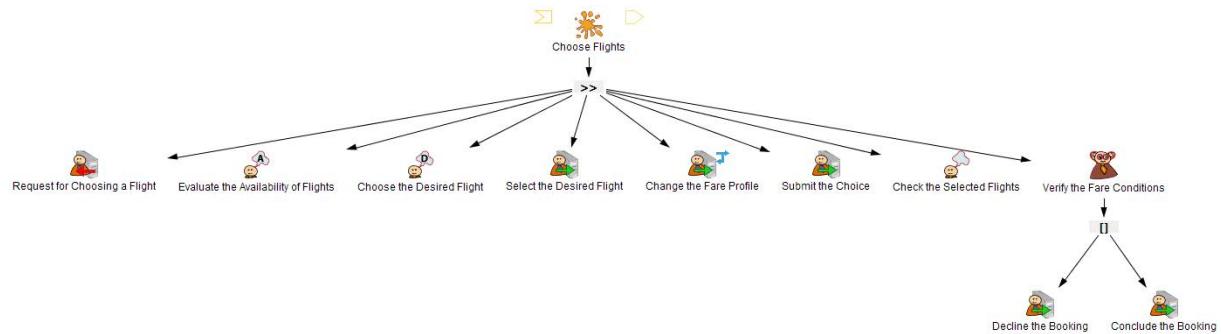


Figure 102. Task Model for Choosing a Flight in Travel Planet.

8.4.1 Extracting Scenarios from the Task Models

Based on the task models presented above, we have extracted 10 scenarios to be tested. The set of 10 scenarios are shown below in Figure 103. The first scenario is intended to book a regular roundtrip (return trip) without including any data, whilst the second one is intended to the same purpose but providing data for the objects values during the execution (data are shown between brackets). The third scenario is intended to book a one-way trip, the fourth one to decline a one-way trip, and the fifth one to book a multidestination trip. Each one of these last five scenarios are accompanied by a similar scenario (presented at the right side of the figure), which does not include the optional tasks, so totalizing the 10 scenarios to be tested.

Scenario 1: Successful Return Trip – Regular Case <ul style="list-style-type: none"> 1 – Go to Book Flights 2 – Inform Departure City 3 – Provide List of Airports 4 – Choose Departure Airport 5 – Inform Arrival City 6 – Provide List of Airports 7 – Choose Arrival Airport 8 – Set Departure Date 9 – Set Departure Time Frame 10 – Set Arrival Date 11 – Set Arrival Time Frame 12 – Choose Number of Passengers 13 – Adjust Timeframe 14 – Select Direct Flights Only 15 – Define Flight Class 16 – Define Companies 	Scenario 6: No Optional Successful Return Trip – Regular Case <ul style="list-style-type: none"> 1 – Go to Book Flights 2 – Inform Departure City 3 – Provide List of Airports 4 – Choose Departure Airport 5 – Inform Arrival City 6 – Provide List of Airports 7 – Choose Arrival Airport 8 – Set Departure Date 9 – Set Arrival Date 10 – Choose Number of Passengers 11 – Submit Search 12 – Present List of Available Flights 13 – Request for Choosing a Flight 14 – Evaluate the Availability of Flights 15 – Choose the Desired Flight 16 – Select the Desired Flight
--	--

17 - Submit Search 18 - Present List of Available Flights 19 - Request for Choosing a Flight 20 - Evaluate the Availability of Flights 21 - Choose the Desired Flight 22 - Select the Desired Flight 23 - Submit the Choice 24 - Check the Selected Flights 25 - Conclude the Booking	17 - Submit the Choice 18 - Check the Selected Flights 19 - Conclude the Booking
Scenario 2: Return Trip With Data 1 - Go to Book Flights 2 - Inform Departure City ("Paris") 3 - Provide List of Airports 4 - Choose Departure Airport ("Paris, Charles-de-Gaulle (CDG)") 5 - Inform Arrival City ("Dallas") 6 - Provide List of Airports 7 - Choose Arrival Airport ("Dallas, Aéroport international de Dallas-Fort Worth (DFW)") 8 - Set Departure Date ("Sam, Déc 1, 2018") 9 - Set Departure Time Frame 10 - Set Arrival Date ("Lun, Déc 10, 2018") 11 - Set Arrival Time Frame 12 - Choose Number of Passengers ("1") 13 - Adjust Timeframe 14 - Select Direct Flights Only 15 - Define Flight Class 16 - Define Companies 17 - Submit Search 18 - Present List of Available Flights 19 - Request for Choosing a Flight 20 - Evaluate the Availability of Flights 21 - Choose the Desired Flight 22 - Select the Desired Flight ("Air France 6111, Air France 6134") 23 - Submit the Choice 24 - Check the Selected Flights 25 - Conclude the Booking	Scenario 7: No Optional Return Trip With Data 1 - Go to Book Flights 2 - Inform Departure City ("Paris") 3 - Provide List of Airports 4 - Choose Departure Airport ("Paris, Charles-de-Gaulle (CDG)") 5 - Inform Arrival City ("Dallas") 6 - Provide List of Airports 7 - Choose Arrival Airport ("Dallas, Aéroport international de Dallas-Fort Worth (DFW)") 8 - Set Departure Date ("Sam, Déc 1, 2018") 9 - Set Arrival Date ("Lun, Déc 10, 2018") 10 - Choose Number of Passengers ("1") 11 - Submit Search 12 - Present List of Available Flights 13 - Request for Choosing a Flight 14 - Evaluate the Availability of Flights 15 - Choose the Desired Flight 16 - Select the Desired Flight ("Air France 6111, Air France 6134") 17 - Submit the Choice 18 - Check the Selected Flights 19 - Conclude the Booking
Scenario 3: Successful One-Way Trip - Regular Case 1 - Go to Book Flights 2 - Inform Departure City 3 - Provide List of Airports 4 - Choose Departure Airport 5 - Inform Arrival City 6 - Provide List of Airports 7 - Choose Arrival Airport 8 - Set Departure Date 9 - Set Departure Time Frame 10 - Choose One-way Trip 11 - Choose Number of Passengers 12 - Adjust Timeframe 13 - Select Direct Flights Only 14 - Define Flight Class 15 - Define Companies 16 - Submit Search 17 - Present List of Available Flights 18 - Request for Choosing a Flight 19 - Evaluate the Availability of Flights 20 - Choose the Desired Flight 21 - Select the Desired Flight 22 - Submit the Choice 23 - Check the Selected Flights 24 - Conclude the Booking	Scenario 8: No Optional Successful One-Way Trip - Regular Case 1 - Go to Book Flights 2 - Inform Departure City 3 - Provide List of Airports 4 - Choose Departure Airport 5 - Inform Arrival City 6 - Provide List of Airports 7 - Choose Arrival Airport 8 - Set Departure Date 9 - Choose One-way Trip 10 - Choose Number of Passengers 11 - Submit Search 12 - Present List of Available Flights 13 - Request for Choosing a Flight 14 - Evaluate the Availability of Flights 15 - Choose the Desired Flight 16 - Select the Desired Flight 17 - Submit the Choice 18 - Check the Selected Flights 19 - Conclude the Booking
Scenario 4: One-Way Trip Declined 1 - Go to Book Flights 2 - Inform Departure City 3 - Provide List of Airports 4 - Choose Departure Airport 5 - Inform Arrival City 6 - Provide List of Airports 7 - Choose Arrival Airport 8 - Set Departure Date 9 - Set Departure Time Frame 10 - Choose One-way Trip 11 - Choose Number of Passengers	Scenario 9: No Optional One-Way Trip Declined 1 - Go to Book Flights 2 - Inform Departure City 3 - Provide List of Airports 4 - Choose Departure Airport 5 - Inform Arrival City 6 - Provide List of Airports 7 - Choose Arrival Airport 8 - Set Departure Date 9 - Choose One-way Trip 10 - Choose Number of Passengers 11 - Submit Search

12 - Adjust Timeframe 13 - Select Direct Flights Only 14 - Define Flight Class 15 - Define Companies 16 - Submit Search 17 - Present List of Available Flights 18 - Request for Choosing a Flight 19 - Evaluate the Availability of Flights 20 - Choose the Desired Flight 21 - Select the Desired Flight 22 - Submit the Choice 23 - Check the Selected Flights 24 - Decline the Booking	12 - Present List of Available Flights 13 - Request for Choosing a Flight 14 - Evaluate the Availability of Flights 15 - Choose the Desired Flight 16 - Select the Desired Flight 17 - Submit the Choice 18 - Check the Selected Flights 19 - Decline the Booking
Scenario 5: Successful Multidestination Trip - Regular Case 1 - Go to Book Flights 2 - Inform Departure City 3 - Provide List of Airports 4 - Choose Departure Airport 5 - Inform Arrival City 6 - Provide List of Airports 7 - Choose Arrival Airport 8 - Set Departure Date 9 - Set Departure Time Frame 10 - Inform Departure City 11 - Provide List of Airports 12 - Choose Departure Airport 13 - Inform Arrival City 14 - Provide List of Airports 15 - Choose Arrival Airport 16 - Set Departure Date 17 - Set Departure Time Frame 18 - Choose Number of Passengers 19 - Adjust Timeframe 20 - Select Direct Flights Only 21 - Define Flight Class 22 - Define Companies 23 - Submit Search 24 - Present List of Available Flights 25 - Request for Choosing a Flight 26 - Evaluate the Availability of Flights 27 - Choose the Desired Flight 28 - Select the Desired Flight 29 - Submit the Choice 30 - Check the Selected Flights 31 - Conclude the Booking	Scenario 10: No Optional Successful Multidestination Trip - Regular Case 1 - Go to Book Flights 2 - Inform Departure City 3 - Provide List of Airports 4 - Choose Departure Airport 5 - Inform Arrival City 6 - Provide List of Airports 7 - Choose Arrival Airport 8 - Set Departure Date 9 - Inform Departure City 10 - Provide List of Airports 11 - Choose Departure Airport 12 - Inform Arrival City 13 - Provide List of Airports 14 - Choose Arrival Airport 15 - Set Departure Date 16 - Choose Number of Passengers 17 - Submit Search 18 - Present List of Available Flights 19 - Request for Choosing a Flight 20 - Evaluate the Availability of Flights 21 - Choose the Desired Flight 22 - Select the Desired Flight 23 - Submit the Choice 24 - Check the Selected Flights 25 - Conclude the Booking

Figure 103. Scenarios extracted to be tested.

8.4.2 Results

According to the testing strategy we presented in chapter 5, testing results are shown in a log indicating, for each step of the User Story scenario, if and where a given step has found an equivalent task in the XML file analyzed, and once it carries an object value associated, which value it is. We have then assessed the task models, based on the set of extracted scenarios presented above (Figure 103). Results of testing for a first complete scenario successfully booking a roundtrip are show hereafter. Such a scenario was obtained by running the scenario “Confirm a Flight Selection” from the User Story with the same name. This scenario calls the scenario “Select a return flight searched without full options” which in turn calls the scenario “Successful Roundtrip Tickets Search”. Corresponding tasks in the scenarios were searched according to the Concept Mapping Table in the appendix of this thesis (appendix A).

Table 25 (and its correspondent chart in Figure 104) brings the results produced by our algorithm when searching for the position of each one of the tasks that composes the scenario. So, the lines of the table (and the legend of the chart) bring the steps in the User Story scenarios, and the columns (and the series of the chart) bring the XML files of the scenarios extracted from the task models. Zeros (0) in the table indicate that a correspondent task for a given step has not

been found in the target file. Values different than zero indicate the position where a correspondent task has been found in the target file. We highlighted in gray at the table which column(s) bring(s) the most suitable target file(s) where the correspondence with the User Story scenario was supposed to be found. For a fully consistent model, it would be necessary that each step in the User Story scenario has found its correspondent task in the same position in the target file. So, in this case, a straight vertical line of points would be seen in the chart below, indicating that a sequential correspondence for each step was found. In the first tested scenario presented above, such a correspondence was supposed to be found in the target file “No Optional Successful Return Trip - Regular Case” once, theoretically, it represents the same user activities in the task model.

8.4.2.1. First Scenario

Scenario: Confirm a Flight Selection	No Optional Return Trip With Data	(Copy) No Optional Successful Multidesti nation Trip - Regular Case	Successful Return Trip - Regular Case	Successful Multidesti nation Trip - Regular Case	(Copy) Successful Multidesti nation Trip - Regular Case	No Optional One-Way Trip Declined	Return Trip With Data	No Optional Successful	Successful One-Way Trip - Regular Case	One-Way Trip Declined	No Optional Successful One-Way Trip - Regular Case
Given I go to "Flight Search"	0	0	0	0	0	0	0	0	0	0	0
When I select "Round Trip"	0	0	0	0	0	0	0	0	0	0	0
And I inform "Toulouse"	0	0	0	0	0	0	0	0	0	0	0
and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	0	0	0	0	0	0	0	0	0	0	0
When I inform "Paris"	0	0	0	0	0	0	0	0	0	0	0
and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	0	0	0	0	0	0	0	0	0	0	0
And I set "Sam, Déc 1, 2018" in the field "Departure Date"	8	8	15	8	16	8	8	8	8	8	8
When I set "Lun, Déc 10, 2018" in the field "Arrival Date"	9	0	10	0	0	10	9	0	0	0	0
And I submit "Search"	11	17	17	23	11	17	11	16	16	16	11
Then will be displayed "2. Sélectionner un voyage"	0	0	0	0	0	0	0	0	0	0	0
Given "Availability Page" is displayed	0	0	0	0	0	0	0	0	0	0	0
When I click on "No Bag" referring to "Air France 7519"	0	0	0	0	0	0	0	0	0	0	0
And I click on "No Bag" referring to "Air France 7522"	0	0	0	0	0	0	0	0	0	0	0
When I click on "Book"	0	0	0	0	0	0	0	0	0	0	0
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0	0	0	0	0	0	0	0	0	0
Given "Confirmation Page" is displayed	0	0	0	0	0	0	0	0	0	0	0
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0	0	0	0	0	0	0	0	0	0
And I click on "Finalize the trip"	0	0	0	0	0	0	0	0	0	0	0
Then will be displayed "Votre voyage a été confirmé!"	0	0	0	0	0	0	0	0	0	0	0

Table 25. Scenario “Confirm a Flight Selection”.

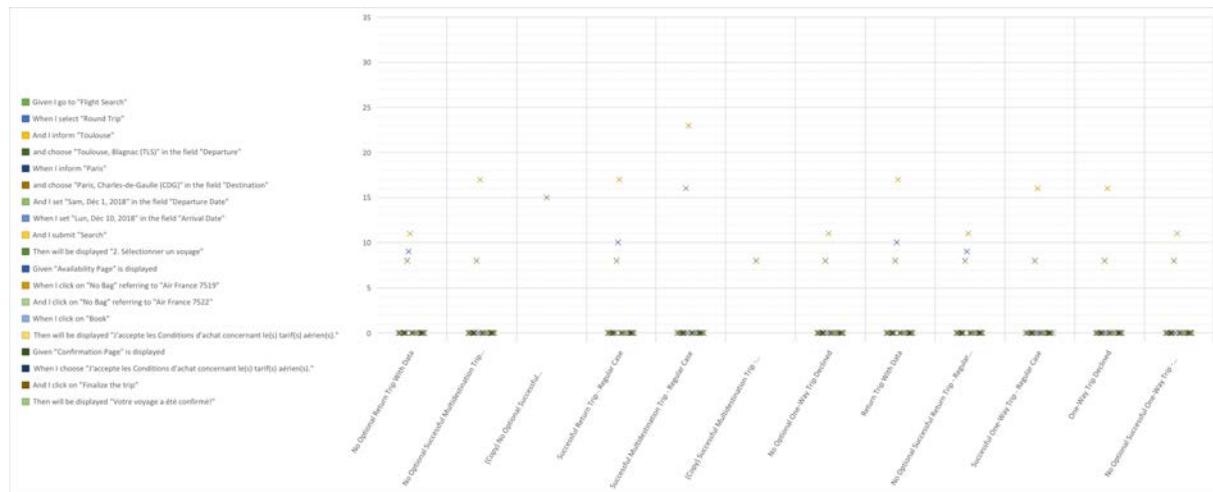


Figure 104. Results of matching; scenario “Confirm a Flight Selection”.

Analyzing the results of this first round of tests, we can notice that most of steps has not found a correspondent task in the target files, i.e. steps in the US scenarios and tasks in the task models are not consistent somehow. The step at the position 1 (“Given I go to ‘Flight Search’”) has not found a correspondent task in any target file because, in the task model, the equivalent task has been modeled as “Go to Book Flights”, so an inconsistency has been found in the name, despite

the position is correct. The step at the position 2 (“When I select ‘Round Trip’”) has not found a correspondent task in any target file due to a different reason. As the interactive task “Choose Round Trip” in the task model (see Figure 100) has been modeled as a parent task of two leave tasks (“Set Arrival Date” and “Set Arrival Time Frame”), when extracting scenarios from such a model, only the leave tasks are kept, so the extracted scenario does never show the interactive task “Choose Round Trip”. Particularly in this case, two other inconsistencies would be found as well: the name of step and task would be different “Select ‘Round Trip’” vs. “Choose ‘Round Trip’”, and the position in which such a task would appear is not the second one, once it is not among the first user tasks according to the model.

Steps at the positions 3-4 and 5-6 concern respectively the set informing/choosing departure and informing/choosing destination. Such tasks have been modeled (and extracted to scenarios) as the triad “Inform Departure City / Provide List of Airports / Choose Departure Airport” and “Inform Arrival City / Provide List of Airports / Choose Arrival Airport”. The intermediate task “Provide List of Airports” (that models the output of the system to the user) has not been modeled in the User Stories, so the step is just composed by the informing and choosing activities. For this reason, such sequence would never find a correspondence in the model, which inevitably would break the forward sequence of tasks in the scenarios. Additionally, another inconsistency that would be identified is that the task model brings tasks named “Inform Departure (Arrival) City” and “Choose Departure (Arrival) Airport”, while the algorithm would search for tasks named “Inform Departure (Destination)” and “Choose Departure (Destination)”.

The step at the position 7 (“And I set ‘Sam, Déc 1, 2018’ in the field ‘Departure Date’”) has found a correspondent task in all the target files, almost always at the position 8. This one-position gap is due to the absence of the task “Choose Round Trip” that was not exported to the scenario as explained above. Besides that, such a step has found two (instead of one) correspondent tasks in the same file. This happened in the target files “Successful Multidestination Trip - Regular Case” and “No Optional Successful Multidestination Trip - Regular Case”, exactly the two ones that describe scenarios for a multidestination trip. As in a multidestination trip, the user must inform at least two flight legs, he/she necessarily needs to inform a “Departure Date” two times, one for each flight leg. That is the reason the algorithm finds the correspondent task “Set Departure Date” two times in these two target files. In the first one, such a task has been found at the positions 8 and 16, and in the second one at the positions 8 and 15. The second occurrence of the task in these files has been marked as “(Copy)” in the table of results presented above. Notice that the associated value informed during the extraction of scenario can also be checked with the value specified in the step. The extracted scenario “Return Trip With Data” in both versions (with and without optional tasks) brings the associated value “Sam, Déc 1, 2018” in the results, that is exactly the same value informed for the correspondent step in the User Story.

The step at the position 8 (“When I set ‘Lun, Déc 10, 2018’ in the field ‘Arrival Date’”) has found a correspondent task at the position 9 in the target files “No Optional Return Trip With Data” and “No Optional Successful Return Trip - Regular Case”, and at the position 10 in the target files “Successful Return Trip - Regular Case” and “Return Trip With Data”. The task “Set Arrival Date” has been found only in those four files because it is only performed in scenarios involving roundtrips (return trips), where an arrival data should be informed. Concerning the position where this task has been found, in the “no-optional” files, it has been found at the position 9 because despite the absence of the task “Choose Round Trip” (which would bring the task “Set Arrival Date” to the position 7), the presence of the two tasks “Provide List of Airports” to inform both departure and destination brings the task “Set Arrival Date” two positions forward, putting it at the position 9. The position 10 in the target files with optional tasks is due to the

presence of the optional task “Set Departure Time Frame” right before the task “Set Arrival Date”.

The step at the position 9 (“And I submit ‘Search’”) has found a correspondent task at different positions in all the target files. The task “Submit Search” has been found at the position 11 in the “no-optional” files (except for the multideestination case that involves informing another flight leg). Looking at the roundtrip case, it highlights an important inconsistency between the scenario presented in the User Story and those extracted from the task model. Apart from the aforementioned absence of the task “Choose Round Trip” and the presence of the two tasks “Provide List of Airports” (which would bring the task “Submit Search” to the position 10), the fact of being found at the position 11 is due to the presence of a previous task named “Choose Number of Passengers” intended to choose the number of passengers that will be included in the booking. This is a mandatory task in the task model but has not been specified as a step in the User Story. It is up to requirements engineers and designers to analyze the models and identify if such a task has been correctly modeled as a mandatory task (so the task model would be correct, and the error would be in the User Stories), or if it is not the case and such a task should be marked as optional in the task model (so the error would be in the task model and not in the User Stories).

Steps from the position 10 until 19 have not found a correspondent task in any target file. At the position 10, it was expected the task “Display 2. Sélectionner un voyage” and the task model brings the task “Present List of Available Flights”. Actually, the task model describes the system task intent which is to present the resulting list of available flights after the search. However, the step in the User Story has opted to specify a given message that would be seen after submitting the search. We can infer that the overall goal of both is the same, but they were specified differently, so there is an inconsistency anyway. At the position 11, it was expected the task “Display Availability Page” and the task model brings the task “Request for Choosing a Flight”. The system action of requesting the user to choose a flight is performed in the availability page, so both tasks could eventually aim at the same purpose, but they are not equivalent once they use different specification strategies. The same occurs with the previous tasks discussed right before.

At the positions 12 and 13, the searched tasks “Click on No Bag” and “Click on No Bag” would find a correspondence with the task “Select the Desired Flight”, but as they specify different behaviors, they cannot be recognized as equivalent. At the position 14, it was expected the task “Click on Book” and the task model brings the task “Submit the Choice”. Due to the use of different semantic behaviors and the lack of context when analyzing only the tasks individually, it is hard to conclude if both tasks intend actually to model the same behavior.

Steps at the positions 15, 16, 17 and 19 do not have tasks modeling the same behaviors in the task model. The searched task “Click on Finalize the trip” at the position 18, just like the ones at the positions from 12 until 14, would find a correspondence with the task “Conclude the Booking” extracted from the task model, however they actually specify different behaviors, so they cannot be recognized as equivalent. Notice finally that the tasks “Evaluate the Availability of Flights”, “Choose the Desired Flight” and “Check the Selected Flights”, both of them included in the scenarios extracted from the task models, are cognitive tasks, so they would not be identifiable by the steps anyway.

Table 26 summarizes the main reasons of failure discussed above for each step of the User Story. Tables (Table 27, Table 28, Table 29 and Table 30) and charts (Figure 105, Figure 106,

Figure 107 and Figure 108) which show the testing results for other scenarios are also presented hereafter. The full log of execution for all scenarios can be found in the appendix B of this thesis.

Step	Main reason of failure
1 - Given I go to 'Flight Search'	Task with different name
2 - When I select 'Round Trip'	Task not extracted to the scenario
3 - And I inform "Toulouse"	Triple and not double sequence of tasks in the task model
4 - ... and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	Triple and not double sequence of tasks in the task model
5 - When I inform 'Paris' ...	Triple and not double sequence of tasks in the task model
6 - ... and choose 'Paris, Charles-de-Gaulle (CDG)' in the field 'Destination'	Triple and not double sequence of tasks in the task model
7 - And I set 'Sam, Déc 1, 2018' in the field 'Departure Date'	Wrong position
8 - When I set 'Lun, Déc 10, 2018' in the field 'Arrival Date'	Wrong position
9 - And I submit 'Search'	Inconsistency between modeling and specification
10 - Then will be displayed '2. Sélectionner un voyage'	Different specification strategy
11 - Given 'Availability Page' is displayed	Different specification strategy
12 - When I click on 'No Bag' referring to 'Air France 7519'	Unpaired behaviors
13 - And I click on 'No Bag' referring to 'Air France 7522'	Unpaired behaviors
14 - When I click on 'Book'	Unpaired behaviors
15 - Then will be displayed 'J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).'	Equivalent behavior missing
16 - Given 'Confirmation Page' is displayed	Equivalent behavior missing
17 - When I choose 'J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).'	Equivalent behavior missing
18 - And I click on 'Finalize the trip'	Unpaired behaviors
19 - Then will be displayed 'Votre voyage a été confirmé!'	Equivalent behavior missing

Table 26. Type of inconsistencies identified in scenarios extracted from task models.

8.4.2.2. Second Scenario

The second scenario “Confirm a Flight Selection (Full Version)” (Table 27, Figure 105) describes the same roundtrip booking but using all the optional fields. Notice that this scenario brings some fixtures for the inconsistency problems identified with the testing of the previous scenario. For example, the first step has been modified to “Given I go to ‘Book Flights’” instead of “Given I go to ‘Flight Search’”, and the step describing the roundtrip selection has been moved forward. Other remarks can be made, notice that as this scenario describes a roundtrip by using the full range of search options, optional steps are never found in the “no optional” target files.

Also notice that despite being found a correspondent task in all the target files, we can see that the step “And I choose the option of value ‘2’ in the field ‘Number of Passengers’” sets the value “2” for the field “Number of Passengers” while in the target file “Return Trip With Data” in its both versions (with and without optional tasks), it has been informed the value “1” during the execution. Considering that values specified for test cases are generally representative of a

data domain that may point out some failure in the system, it is important to look carefully at such kind of inconsistency in the assessed artifacts.

Scenario: Confirm a Flight Selection (Full Version)	No Optional Return Trip With Data	(Copy) No Optional Multidestination Trip - Regular Case	Successful Return Trip - Regular Case	Successful Multidestination Trip - Regular Case	(Copy) Successful Multidestination Trip - Regular Case	No Optional One-Way Trip Declined	Return Trip With Data	No Optional Successful Return Trip - Regular Case	Successful One-Way Trip - Regular Case	One-Way Trip Declined	No Optional Successful One-Way Trip - Regular Case
Given I go to "Book Flights"	1	1		1	1		1	1	1	1	1
When I inform "Toulouse"	0	0		0	0		0	0	0	0	0
and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	0	0		0	0		0	0	0	0	0
And I inform "Paris"	0	0		0	0		0	0	0	0	0
and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	0	0		0	0		0	0	0	0	0
When I set "Sam, Déc 1, 2018" in the field "Departure Date"	8	8	15	8	16	8	8	8	8	8	8
And I set "08:00" in the field "Departure Time Frame"	0	0		9	17	9	0	9	9	9	0
When I choose "Round Trip"	0	0		0	0		0	0	0	0	0
And I set "Sun, Déc 10, 2018" in the field "Arrival Date"	9	0		10	0		0	10	9	0	0
When I set "10:00" in the field "Arrival Time Frame"	0	0		11	0		0	11	0	0	0
And I choose the option of value "2" in the field "Number of Passengers"	10	16		12	18		10	12	10	11	10
When I set "6" in the field "Timeline"	0	0		0	0		0	0	0	0	0
And I select "Direct Flights Only"	0	0		14	20		0	14	0	13	13
When I choose the option of value "Economique" in the field "Flight Class"	0	0		0	0		0	0	0	0	0
And I set "Air France" in the field "Companies"	0	0		0	0		0	0	0	0	0
When I submit "Search"	11	17		17	23		11	17	11	16	16
Then will be displayed "2. Sélectionner un voyage"	0	0		0	0		0	0	0	0	0
Given "Availability Page" is displayed	0	0		0	0		0	0	0	0	0
When I click on "No Bag" referring to "Air France 7519"	0	0		0	0		0	0	0	0	0
And I click on "No Bag" referring to "Air France 7522"	0	0		0	0		0	0	0	0	0
When I click on "Book"	0	0		0	0		0	0	0	0	0
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0		0	0		0	0	0	0	0
Given "Confirmation Page" is displayed	0	0		0	0		0	0	0	0	0
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0		0	0		0	0	0	0	0
And I click on "Finalize the trip"	0	0		0	0		0	0	0	0	0
Then will be displayed "Votre voyage a été confirmé!"	0	0		0	0		0	0	0	0	0

Table 27. Scenario "Confirm a Flight Selection (Full Version)".

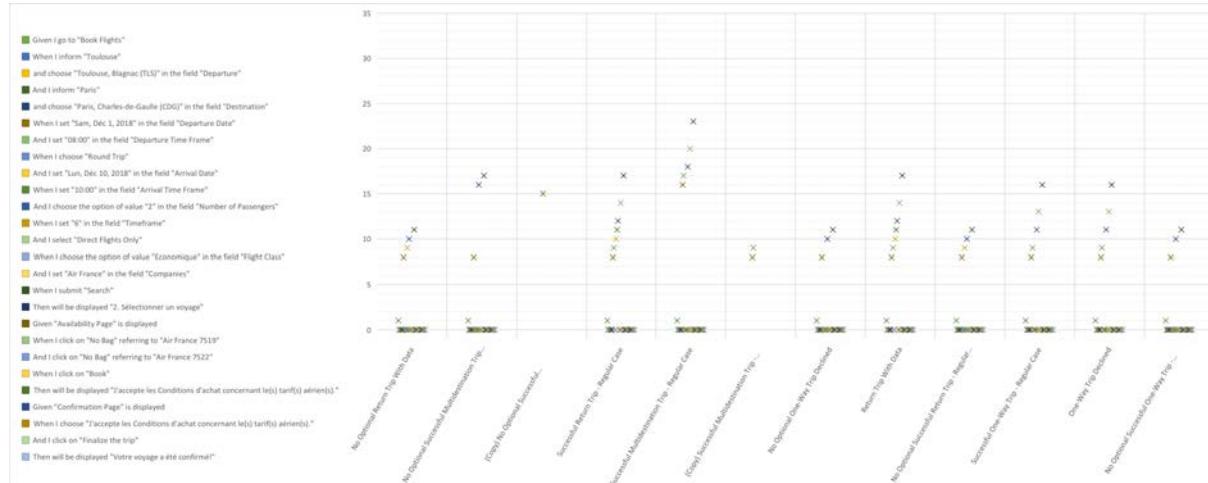


Figure 105. Results of matching: scenario "Confirm a Flight Selection (Full Version)".

8.4.2.3. Other Scenarios

Below are presented the results for other assessed scenarios, including scenarios to confirm and decline a one-way trip, and confirm a multidestination trip. As discussed in chapter 5, as task models are not designed to model user's errors, scenarios from the User Stories which test error situations were not assessed with the extracted task model scenarios. As user errors are not part of a user goal, they are usually omitted from tasks descriptions, making this kind of test fail. Means of representing these potential errors on task models is being recently studied (Fahssi, Martinie and Palanque, 2015). Once it is implemented in the model, tests could run using the same approach to identify this kind of error.

Scenario: Confirm a Flight Selection for a One-Way Trip	No Optional Return Trip With Data	(Copy) No Optional Multidestination Trip - Regular Case	Successful Return Trip - Regular Case	(Copy) Successful Multidestination Trip - Regular Case	No Optional One-Way Trip Declined	Return Trip With Data	No Optional Successful Return Trip - Regular Case	Successful One-Way Trip - Regular Case	One-Way Trip Declined	No Optional Successful One-Way Trip - Regular Case
Given I go to "Book Flights"	1	1	1	1	1	1	1	1	1	1
When I inform "Toulouse"	0	0	0	0	0	0	0	0	0	0
and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	0	0	0	0	0	0	0	0	0	0
And I inform "Paris"	0	0	0	0	0	0	0	0	0	0
and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	0	0	0	0	0	0	0	0	0	0
When I set "Sam, Déc 1, 2018" in the field "Departure Date"	8	8	15	8	16	8	8	8	8	8
And I choose "One-way Trip"	0	0	0	0	0	9	0	0	10	9
When I submit "Search"	11	17	17	23	11	17	11	16	16	11
Then will be displayed "2. Sélectionner un voyage"	0	0	0	0	0	0	0	0	0	0
Given "Availability Page" is displayed	0	0	0	0	0	0	0	0	0	0
When I click on "No Bag" referring to "Air France 7519"	0	0	0	0	0	0	0	0	0	0
And I click on "Book"	0	0	0	0	0	0	0	0	0	0
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0	0	0	0	0	0	0	0	0
Given "Confirmation Page" is displayed	0	0	0	0	0	0	0	0	0	0
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0	0	0	0	0	0	0	0	0
And I click on "Finalize the trip"	0	0	0	0	0	0	0	0	0	0
Then will be displayed "Votre voyage a été confirmé!"	0	0	0	0	0	0	0	0	0	0

Table 28. Scenario "Confirm a Flight Selection for a One-Way Trip".

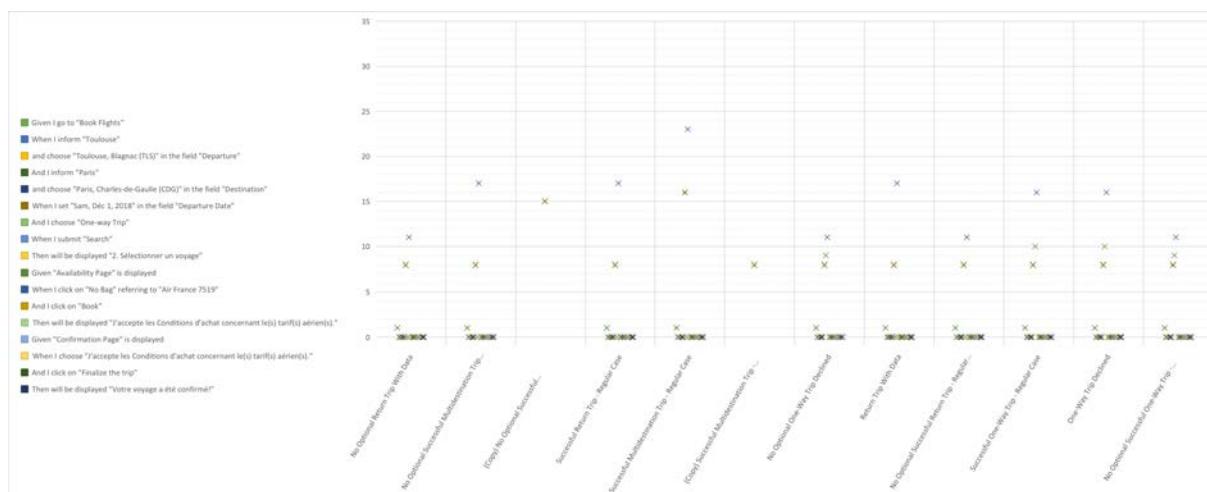


Figure 106. Results of matching: scenario "Confirm a Flight Selection for a One-Way Trip".

Scenario: Confirm a Flight Selection for a Multidestination Trip	No Optional Return Trip With Data	No Optional Successful Multidestination Trip - Regular Case	(Copy) No Optional Successful Multidestination Trip - Regular Case	Successful Return Trip - Regular Case	(Copy) Successful Multidestination Trip - Regular Case	No Optional One-Way Trip Declined	Return Trip With Data	No Optional Successful Return Trip - Regular Case	Successful One-Way Trip - Regular Case	One-Way Trip Declined	No Optional Successful One-Way Trip - Regular Case
Given I go to "Book Flights"	1	1	1	1	1	1	1	1	1	1	1
When I choose "Multidestination Trip"	0	0	0	0	0	0	0	0	0	0	0
And I inform "Toulouse"	0	0	0	0	0	0	0	0	0	0	0
and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	0	0	0	0	0	0	0	0	0	0	0
When I inform "Paris"	0	0	0	0	0	0	0	0	0	0	0
and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	0	0	0	0	0	0	0	0	0	0	0
And I set "Sam, Déc 1, 2018" in the field "Departure Date"	8	8	15	8	16	8	8	8	8	8	8
When I inform "Paris"	0	0	0	0	0	0	0	0	0	0	0
and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Departure"	0	0	0	0	0	0	0	0	0	0	0
And I inform "Nice"	0	0	0	0	0	0	0	0	0	0	0
and choose "Nice, Côte d'Azur (NCE)" in the field "Destination"	0	0	0	0	0	0	0	0	0	0	0
When I set "Sam, Déc 10, 2018" in the field "Departure Date"	8	8	15	8	16	8	8	8	8	8	8
And I submit "Search"	11	17	17	23	11	17	11	16	16	11	11
Then will be displayed "2. Sélectionner un voyage"	0	0	0	0	0	0	0	0	0	0	0
Given "Availability Page" is displayed	0	0	0	0	0	0	0	0	0	0	0
When I click on "No Bag" referring to "Air France 7519"	0	0	0	0	0	0	0	0	0	0	0
And I click on "Book"	0	0	0	0	0	0	0	0	0	0	0
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0	0	0	0	0	0	0	0	0	0
Given "Confirmation Page" is displayed	0	0	0	0	0	0	0	0	0	0	0
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0	0	0	0	0	0	0	0	0	0
And I click on "Finalize the trip"	0	0	0	0	0	0	0	0	0	0	0
Then will be displayed "Votre voyage a été confirmé!"	0	0	0	0	0	0	0	0	0	0	0

Table 29. Scenario "Confirm a Flight Selection for a Multidestination Trip".

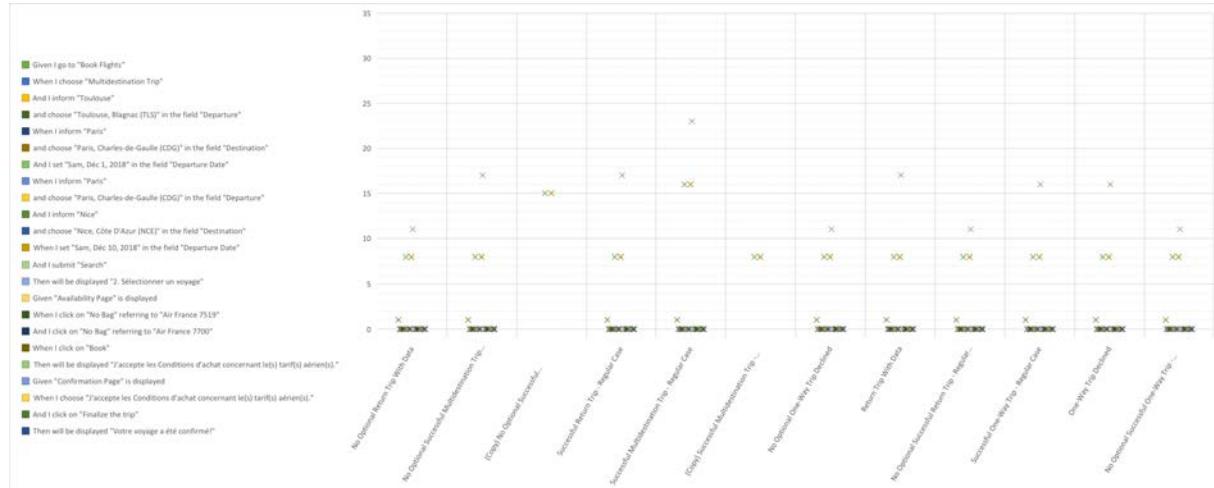


Figure 107. Results of matching: scenario “Confirm a Flight Selection for a Multidestination Trip”.

Scenario: Decline a Flight Selection	No Optional Return Trip With Data	No Optional Successful Multidestination Trip - Regular Case	(Copy) No Optional Successful Multidestination Trip - Regular Case	Successful Return Trip - Regular Case	Successful Multidestination Trip - Regular Case	(Copy) Successful Multidestination Trip - Regular Case	No Optional One-Way Trip Declined	Return Trip With Data	No Optional Successful One-Way Trip - Regular Case	Successful One-Way Trip - Regular Case	One-Way Trip Declined	No Optional Successful One-Way Trip - Regular Case
Given I go to "Book Flights"	1	1		1	1		1	1	1	1	1	1
When I inform "Toulouse"	0	0		0	0		0	0	0	0	0	0
and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	0	0		0	0		0	0	0	0	0	0
And I inform "Paris"	0	0		0	0		0	0	0	0	0	0
and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	0	0		0	0		0	0	0	0	0	0
When I set "Sam, Déc 1, 2018" in the field "Departure Date"	8	8	15	8	16	8	8	8	8	8	8	8
And I choose "One-way Trip"	0	0		0	0		9	0	0	10	10	9
When I submit "Search"	11	17		17	23		11	17	11	16	16	11
Then will be displayed "2. Sélectionner un voyage"	0	0		0	0		0	0	0	0	0	0
Given "Availability Page" is displayed	0	0		0	0		0	0	0	0	0	0
When I click on "No Bag" referring to "Air France 7519"	0	0		0	0		0	0	0	0	0	0
And I click on "Book"	0	0		0	0		0	0	0	0	0	0
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0		0	0		0	0	0	0	0	0
Given "Confirmation Page" is displayed	0	0		0	0		0	0	0	0	0	0
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	0	0		0	0		0	0	0	0	0	0
And I click on "Decline the trip"	0	0		0	0		0	0	0	0	0	0
Then will be displayed "Votre voyage a été annulé!"	0	0		0	0		0	0	0	0	0	0

Table 30. Scenario “Decline a Flight Selection”.

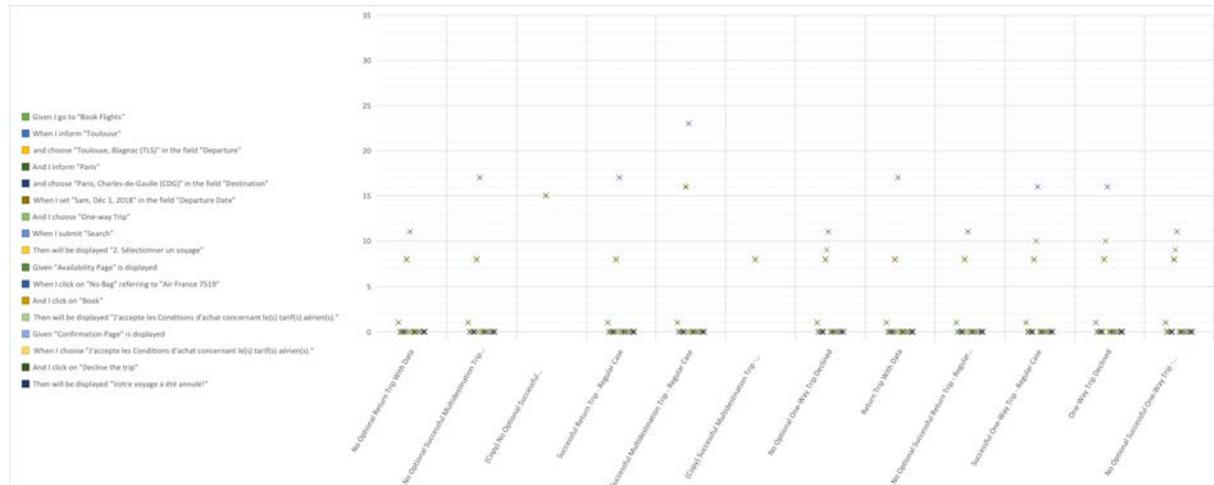


Figure 108. Results of matching: scenario “Decline a Flight Selection”.

8.4.3 Types of Inconsistencies Identified

By summarizing the results presented above, below we formalize the types of inconsistencies found by our testing approach when assessing the task models:

- **Task with different names**, refers to tasks that are present both in the task model and in the User Story scenario but written with a different name.
- **Task not extracted to the scenario**, refers to tasks that are effectively modeled in the task model but, due to the type of operators used or the presence (or not) of other refined tasks after them in the model, causes that, during the extraction process, such tasks do not be taken to the extracted scenarios.
- **Different number of sequences of tasks in the task model**, occurs when there are more tasks in the task model scenario than steps in the User Story scenario to accomplish the same behavior.
- **Wrong position**, which is related to tasks that are found in different positions than their equivalent steps in User Stories.
- **Conflict between specification and modeling**, refers to tasks modeled in the task model (and consequently exported to its scenarios) that are not present the requirements specification in the User Stories.
- **Different specification strategies**, refers to the specification of behaviors that could eventually aim at the same purpose, but were specified using different strategies, i.e. requiring to perform (or verify) different actions.
- **Unpaired behaviors**, refers to tasks that would find a correspondence with the steps in the User Stories, but as they actually specify different behaviors, they cannot be recognized as such.
- **Equivalent behaviors missing**, refers to behaviors that are really missing in the extracted task model scenario, like steps that are present in the User Story, but cannot find correspondent tasks in the task model.

8.5. Modeling and Assessing UI Prototypes

UI prototypes for this case study have been developed using Balsamiq Mockups. The sequence of figures in Table 31 shows the different states and designs of the developed prototypes. Figure 109 illustrates our first approach for a UI prototype to search flights. The figure designs a UI for searching flights based on a round trip (and Figure 118 based on a one-way/multidestination trip). On the right side (Figure 110), we present a changed UI redesigned to fix the problems found during the batteries of tests. Figure 111 (and its redesign in Figure 112) illustrates the next UI in sequence, showing the list of flights matching the selection criteria. When the user selects one of the available flights, then the system turns out to the state shown in Figure 119. The user, at this state, can confirm his/her selection or change the fare profile of his/her flight.

Figure 113 (redesigned in Figure 114) finally shows screens of confirmation of a flight selection. On the prototype presented, the user can accept the general terms and conditions and confirm his/her booking or withdraw his/her trip. In such a case, the system asks the user to confirm his/her choice (Figure 116), and if confirmed, cancel the trip (Figure 117). If the user does not confirm the withdrawing or opt to confirm the trip at the first stage, then the system shows a message confirming the book has been taken into account (Figure 115).

Unlike the assessing of task models where we parse all the steps at once, to assess the UI prototypes, we parse each step at a time. It means that if an error is found in a given step, the test stops until it has been fixed. To discuss the results that we got by testing different versions of Balsamiq prototypes, we present hereafter results of several batteries of testing in each version of the prototypes developed to perform a successful roundtrip booking. We present sequentially each step of the target scenario, the correspondent extracts of interaction elements in the

Balsamiq XML file, errors that have been found in a given battery, the solution proposed to fix them, and finally the subsequent battery of tests following the fixes. Once the goal is to assess the most possible number of interaction elements in the prototype, we have chosen to run our tests presented below on the full versions of the scenarios, i.e. in those ones interacting with all the optional fields.

We notice that the first battery of tests found an error already in the first step (“Given I go to ‘Book Flights’”). It was expected a correspondent element “BrowserWindow” associated to the name “Book Flights” in the prototype, but the element found was a “SubTitle”. The “BrowserWindow” was named “Travel Planet”, the name of the system under testing. As the behavior “goTo” is supposed to be performed only in a window (and its variants), such a step could not be performed in a field describing a subtitle, which is a semantically inconsistent file for that behavior. Actually, at this point, the designer realized that “Book Flights” was not a good name for a window, once it refers to the whole process of booking a trip, and not only to the window for searching flights specifically. As a solution to fix it, the window was named “Flight Search” and both the scenario and the prototype have been updated.

In the second battery of tests, the steps 1, 2 and 3 passed, and an error was found in the step 4 (“When I set ‘Sam, Déc 1, 2018’ in the field ‘Departure Date’”). This error refers to the label “Departure Date” that has been found in a different group than the element “DataChooser” which was used to model it. As detailed in chapter 6, Balsamiq models elements either as independent instances (i.e. with the name and the interaction element defined in the same tag), or as part of a group (i.e. defining the name in the tag “label” and the interaction element itself in another tag). In the second case, the group must be modeled as a single unit, with a unique identifier. The label “Departure Date” was found in a given group and its interaction element “DataChooser” in another one, so they could not be recognized as a single unit. To fix the error, they were regrouped.

In the third battery of tests, the steps 4, 5 and 6 passed, and the same error was found in the step 7 (And I set ‘Lun, Déc 10, 2018’ in the field ‘Arrival Date’”). The label “Arrival Date” and its correspondent element “DataChooser” were found in different groups. The same solution to fix it was applied. In the fourth battery of tests, the steps 7 and 8 passed, and an error was found in the step 9 (And I choose the option of value ‘2’ in the field ‘Number of Passengers’”). The field “Number of Passengers” was not found in the prototype. It was added to fix the error.

This screenshot shows the initial UI prototype for searching flights. It features a header 'Travel Planet' and a title 'Book Flights'. A radio button group for 'Round trip' or 'One-way / Multidestination' is present. Below this are fields for 'From' and 'To' locations, and date pickers for 'Departure Date' and 'Arrival Date'. There are dropdown menus for 'Time Frame' and 'Only direct flights'. An 'Advanced Search' section includes a 'Class' dropdown and a 'Companies' search bar with three input fields. A 'Search' button is at the bottom.

Figure 109. UI prototype for searching flights (first version).

This screenshot shows the revised UI prototype. The title has changed to 'Flight Search'. The 'Round trip' radio button is selected. The 'From' and 'To' fields are labeled 'Departure' and 'Destination' respectively. The date pickers are labeled 'Departure Date' and 'Arrival Date'. The 'Timeframe' dropdown has been renamed 'Timeline'. The 'Companies' section now includes a dropdown for 'Flight Class' and three separate search fields for 'Company 1', 'Company 2', and 'Company 3'. A 'Direct Flights Only' checkbox is also present.

Figure 110. UI prototype for searching flights (revised version after testing).

Scenario: Successful Roundtrip Tickets Search With Full Options				
Battery	Step	Balsamiq extract (XML source file)	Error	
1	Given I go to “Book Flights” (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	<control controlID="2" controlTypeID="com.balsamiq.mockups::SubTitle" x="588" y="244" w="-1" h="-1" measuredW="133" measuredH="27" zOrder="2" locked="false" isInGroup="-1"><controlProperties><text>Book%20Flights</text></controlProperties></control>	Expected “BrowserWindow”, but the element was “SubTitle”.	
2	Given I go to “Flight Search”	<control controlID="0" controlTypeID="com.balsamiq.mockups::BrowserWindow" x="567" y="146" w="651" h="566" measuredW="450" measuredH="400" zOrder="0" locked="false" isInGroup="-1"><controlProperties><text>Flight%20Search</text></controlProperties></control>	-	

	<p>When I inform “Toulouse” and choose “Toulouse, Blagnac (TLS)” in the field “Departure”</p>	<pre><control controlID="0" controlTypeID="com.balsamiq.mockups::SearchBox" x="0" y="21" w="277" h="-1" measuredW="120" measuredH="25" zOrder="0" locked="false" isInGroup="17"> <controlProperties> <text>From</text> </controlProperties> </control> <control controlID="1" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="88" h="-1" measuredW="60" measuredH="21" zOrder="1" locked="false" isInGroup="17"> <controlProperties> <text>Departure</text> </controlProperties> </control></pre>	
	<p>And I inform “Paris” and choose “Paris, Charles-de-Gaulle (CDG)” in the field “Destination”</p>	<pre><control controlID="0" controlTypeID="com.balsamiq.mockups::SearchBox" x="0" y="21" w="277" h="-1" measuredW="120" measuredH="25" zOrder="0" locked="false" isInGroup="18"> <controlProperties> <text>To</text> </controlProperties> </control> <control controlID="1" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="98" h="-1" measuredW="67" measuredH="21" zOrder="1" locked="false" isInGroup="18"> <controlProperties> <text>Destination</text> </controlProperties> </control></pre>	
	<p>When I set “Sam, Déc 1, 2018” in the field “Departure Date” (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)</p>	<pre><control controlID="0" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="-1" h="-1" measuredW="92" measuredH="21" zOrder="0" locked="false" isInGroup="0"> <controlProperties> <text>Departure%20Date</text> </controlProperties> </control> ... <control controlID="1" controlTypeID="com.balsamiq.mockups::DateChooser" x="0" y="21" w="-1" h="-1" measuredW="90" measuredH="25" zOrder="1" locked="false" isInGroup="22"> <controlProperties> <text>%20%20/%202020/%20%20%20</text> </controlProperties> </control></pre>	The label “Departure Date” and the element “DataChooser” are in different groups.
3	<p>When I set “Sam, Déc 1, 2018” in the field “Departure Date”</p>	<pre><control controlID="0" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="-1" h="-1" measuredW="92" measuredH="21" zOrder="0" locked="false" isInGroup="38"> <controlProperties> <text>Departure%20Date</text> </controlProperties> </control> <control controlID="1" controlTypeID="com.balsamiq.mockups::DateChooser" x="0" y="21" w="-1" h="-1" measuredW="90" measuredH="25" zOrder="1" locked="false" isInGroup="38"> <controlProperties> <text>%20%20/%20%20/%20%20%20</text> </controlProperties> </control></pre>	
	<p>And I set “08:00” in the field “Departure Time Frame”</p>	<pre><control controlID="24" controlTypeID="com.balsamiq.mockups::ComboBox" x="702" y="396" w="-1" h="-1" measuredW="163" measuredH="24" zOrder="6" locked="false" isInGroup="-1"> <controlProperties> <text>Departure%20Time%20Frame</text> </controlProperties> </control></pre>	

		</controlProperties> </control>	
	When I choose “Round Trip”	<control controlID="0" controlTypeID="com.balsamiq.mockups::RadioButton" x="-0" y="0" w="-1" h="-1" measuredW="77" measuredH="22" zOrder="0" locked="false" isInGroup="32"> <controlProperties> <state>selected</state> <text>Round%20trip</text> </controlProperties> </control>	-
	And I set “Lun, Déc 10, 2018” in the field “Arrival Date” (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	<control controlID="0" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="-1" h="-1" measuredW="69" measuredH="21" zOrder="0" locked="false" isInGroup="0"> <controlProperties> <text>Arrival%20Date</text> </controlProperties> </control> ... <control controlID="1" controlTypeID="com.balsamiq.mockups::DateChooser" x="0" y="21" w="-1" h="-1" measuredW="90" measuredH="25" zOrder="1" locked="false" isInGroup="23"> <controlProperties> <text>%20%20/%20%20/%20%20%20</text> </controlProperties> </control>	The label “Arrival Date” and the element “DataChooser” are in different groups.
4	And I set “Lun, Déc 10, 2018” in the field “Arrival Date”	<control controlID="0" controlTypeID="com.balsamiq.mockups::DateChooser" x="0" y="21" w="-1" h="-1" measuredW="90" measuredH="25" zOrder="0" locked="false" isInGroup="41"> <controlProperties> <text>%20%20/%20%20/%20%20%20</text> </controlProperties> </control> <control controlID="1" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="-1" h="-1" measuredW="69" measuredH="21" zOrder="1" locked="false" isInGroup="41"> <controlProperties> <text>Arrival%20Date</text> </controlProperties> </control>	-
	When I set “10:00” in the field “Arrival Time Frame”	<control controlID="25" controlTypeID="com.balsamiq.mockups::ComboBox" x="1048" y="396" w="-1" h="-1" measuredW="141" measuredH="24" zOrder="7" locked="false" isInGroup="-1"> <controlProperties> <text>Arrival%20Time%20Frame</text> </controlProperties> </control>	-
	And I choose the option of value “2” in the field “Number of Passengers” (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	-	The field “Number of Passengers” does not exist.
5	And I choose the option of value “2” in the field “Number of Passengers”	<control controlID="0" controlTypeID="com.balsamiq.mockups::ComboBox" x="151" y="0" w="-1" h="-1" measuredW="36" measuredH="24" zOrder="0" locked="false" isInGroup="44"> <controlProperties> <text>1</text> </controlProperties> </control>	-

		<pre><control controlID="1" controlTypeID="com.balsamiq.mockups::Label" x="0" y="2" w="-1" h="-1" measuredW="136" measuredH="21" zOrder="1" locked="false" isInGroup="44"> <controlProperties> <text>Number%20of%20Passengers</text> </controlProperties> </control></pre>	
	When I set "6" in the field "Timeframe" (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	<pre><control controlID="28" controlTypeID="com.balsamiq.mockups::ComboBox" x="588" y="478" w="-1" h="-1" measuredW="100" measuredH="24" zOrder="9" locked="false" isInGroup="-1"> <controlProperties> <text>Time%20Frame</text> </controlProperties> </control></pre>	Expected field "Timeframe" but was "Time Frame".
6	When I set "6" in the field "Timeframe"	<pre><control controlID="28" controlTypeID="com.balsamiq.mockups::ComboBox" x="588" y="478" w="-1" h="-1" measuredW="100" measuredH="24" zOrder="9" locked="false" isInGroup="-1"> <controlProperties> <text>Timeframe</text> </controlProperties> </control></pre>	-
	And I select "Direct Flights Only" (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	<pre><control controlID="33" controlTypeID="com.balsamiq.mockups::CheckBox" x="912" y="479" w="-1" h="-1" measuredW="125" measuredH="22" zOrder="11" locked="false" isInGroup="-1"> <controlProperties> <text>Only%20direct%20flights</text> </controlProperties> </control></pre>	Expected field "Direct Flights Only" but was "Only direct flights".
7	And I select "Direct Flights Only"	<pre><control controlID="33" controlTypeID="com.balsamiq.mockups::CheckBox" x="912" y="479" w="-1" h="-1" measuredW="125" measuredH="22" zOrder="11" locked="false" isInGroup="-1"> <controlProperties> <text>Direct%20Flights%20Only</text> </controlProperties> </control></pre>	-
	When I choose the option of value "Economique" in the field "Flight Class" (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	<pre><control controlID="35" controlTypeID="com.balsamiq.mockups::ComboBox" x="606" y="552" w="-1" h="-1" measuredW="64" measuredH="24" zOrder="12" locked="false" isInGroup="-1"> <controlProperties> <text>Class</text> </controlProperties> </control></pre>	Expected field "Flights Class" but was "Class".
8	When I choose the option of value "Economique" in the field "Flight Class"	<pre><control controlID="35" controlTypeID="com.balsamiq.mockups::ComboBox" x="606" y="552" w="-1" h="-1" measuredW="64" measuredH="24" zOrder="12" locked="false" isInGroup="-1"> <controlProperties> <text>Flight%20Class</text> </controlProperties> </control></pre>	-
	And I set "Air France" in the field "Companies" (FAILED) (java.lang.AssertionError: expected:<1> but was:<3>)	<pre><control controlID="0" controlTypeID="com.balsamiq.mockups::SearchBox" x="0" y="21" w="67" h="-1" measuredW="120" measuredH="24" zOrder="0" locked="false" isInGroup="27"> <controlProperties> <text/> </controlProperties> </control></pre> <pre><control controlID="1" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="88" h="-1" measuredW="67" measuredH="21" zOrder="1" locked="false" isInGroup="27"></pre>	Three elements "SearchBox" to address the same field "Companies".

		<pre> <controlProperties> <text>Companies</text> </controlProperties> </control> <control controlID="2" controlTypeID="com.balsamiq.mockups::SearchBox" x="81" y="21" w="67" h="-1" measuredW="120" measuredH="24" zOrder="2" locked="false" isInGroup="27"> <controlProperties> <text/> </controlProperties> </control> <control controlID="3" controlTypeID="com.balsamiq.mockups::SearchBox" x="164" y="21" w="67" h="-1" measuredW="120" measuredH="24" zOrder="3" locked="false" isInGroup="27"> <controlProperties> <text/> </controlProperties> </control> </pre>	
9	<p>And I set “Air France” in the field “Company 1”</p>	<pre> <control controlID="0" controlTypeID="com.balsamiq.mockups::SearchBox" x="0" y="21" w="67" h="-1" measuredW="120" measuredH="24" zOrder="0" locked="false" isInGroup="27"> <controlProperties> <text>Company#201</text> </controlProperties> </control> <control controlID="1" controlTypeID="com.balsamiq.mockups::Label" x="0" y="0" w="88" h="-1" measuredW="67" measuredH="21" zOrder="1" locked="false" isInGroup="27"> <controlProperties> <text>Companies</text> </controlProperties> </control> <control controlID="2" controlTypeID="com.balsamiq.mockups::SearchBox" x="81" y="21" w="67" h="-1" measuredW="120" measuredH="24" zOrder="2" locked="false" isInGroup="27"> <controlProperties> <text>Company#202</text> </controlProperties> </control> <control controlID="3" controlTypeID="com.balsamiq.mockups::SearchBox" x="164" y="21" w="67" h="-1" measuredW="120" measuredH="24" zOrder="3" locked="false" isInGroup="27"> <controlProperties> <text>Company#203</text> </controlProperties> </control> </pre>	-
	<p>When I submit “Search”</p>	<pre> <control controlID="14" controlTypeID="com.balsamiq.mockups::Button" x="1126" y="678" w="-1" h="-1" measuredW="63" measuredH="27" zOrder="5" locked="false" isInGroup="1"> <controlProperties> <text>Search</text> </controlProperties> </control> </pre>	-
	<p>Then will be displayed “2. Sélectionner un voyage” (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)</p>	-	Dynamic behavior between screens. Untraceable interaction.

This screenshot shows a 'Book Flights' page from 'Travel Planet'. At the top, there's a navigation bar with back, forward, and search icons. Below it, a title 'Book Flights' and a 'Back' button. A section titled 'Choose your flight below:' contains a table with flight information. The table has columns for Flight, Discount, Classic, and Flex prices. The first row is highlighted in grey. At the bottom, there are fields for 'Your selection:' and 'Price:', and a 'Book' button.

Flight	Discount	Classic	Flex
AA2512 Paris CDG (08:12) - London LHR (09:00)	90.00 €	150.00 €	280.00 €
AF2591 Paris ORY (08:35) - London LGW (09:15)	115.00 €	210.00 €	350.00 €
AF2348 Paris CDG (09:20) - London LHR (10:10)	-	170.00 €	300.00 €
AF2216 Paris CDG (10:40) - London LGW (11:22)	95.00 €	160.00 €	290.00 €
AF1544 Paris ORY (12:00) - London LGW (12:50)	-	-	410.00 €
AF3551 Paris CDG (13:30) - London LHR (14:20)	70.00 €	120.00 €	210.00 €

Figure 111. UI prototype for choosing flights (first version).

This screenshot shows the same 'Book Flights' page after testing. The layout is identical to Figure 111, but the data in the table has been modified. The first row is no longer highlighted in grey, and the 'No Bag' column has been added to the table header. The 'Book' button remains at the bottom.

Flight	No Bag	Classic	Flex
AA2512 Paris CDG (08:12) - London LHR (09:00)	90.00 €	150.00 €	280.00 €
AF2591 Paris ORY (08:35) - London LGW (09:15)	115.00 €	210.00 €	350.00 €
AF2348 Paris CDG (09:20) - London LHR (10:10)	-	170.00 €	300.00 €
AF2216 Paris CDG (10:40) - London LGW (11:22)	95.00 €	160.00 €	290.00 €
AF1544 Paris ORY (12:00) - London LGW (12:50)	-	-	410.00 €
AF3551 Paris CDG (13:30) - London LHR (14:20)	70.00 €	120.00 €	210.00 €

Figure 112. UI prototype for choosing flights (revised version after testing).

Scenario: Select a Return Flight Searched With Full Options			
Battery	Step	Balsamiq extract (XML source file)	Error
1	Given "Availability Page" is displayed (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	-	"Availability Page" does not exist.
2	Given "Availability Page" is displayed	<pre><control controlID="0" controlTypeID="com.balsamiq.mockups::BrowserWindow" x="567" y="146" w="651" h="622" measuredW="450" measuredH="400" zOrder="0" locked="false" isInGroup="-1"> <controlProperties> <text>Availability%20Page</text> </controlProperties> </control></pre>	-
	When I click on "No Bag" referring to "Air France 7519" (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	<pre><control controlID="27" controlTypeID="com.balsamiq.mockups::DataGrid" x="607" y="346" w="570" h="219" measuredW="518" measuredH="219" zOrder="3" locked="false" isInGroup="-1"> <controlProperties> <align>center</align> <borderStyle>none</borderStyle> <hLines>false</hLines></pre>	Expected field "No Bag" but was "Discount".

		<pre><map>%3Carea%20shape%3D%22rect%22%20coords%3D%22366...</map> <rowHeight>33</rowHeight> <text>Flight%2C%20Discount%2C%20Classic%2C%20Flex%...</text> <value>15</value> <verticalScrollbar>true</verticalScrollbar> <vLines>true</vLines> </controlProperties> </control></pre>	
3	When I click on “ No Bag ” referring to “Air France 7519”	<pre><control controlID="27" controlTypeID="com.balsamiq.mockups::DataGrid" x="607" y="346" w="570" h="219" measuredW="516" measuredH="219" zOrder="3" locked="false" isInGroup="-1"> <controlProperties> <align>center</align> <borderStyle>none</borderStyle> <hLines>false</hLines> <map>%3Carea%20shape%3D%22rect%22%20coords%3D%22366...</map> <rowHeight>33</rowHeight> <text>Flight%2C%20No%20Bag%2C%20Classic%2C%20Flex%...</text> <value>15</value> <verticalScrollbar>true</verticalScrollbar> <vLines>true</vLines> </controlProperties> </control></pre>	-
	And I click on “ No Bag ” referring to “Air France 7522”	<pre><control controlID="27" controlTypeID="com.balsamiq.mockups::DataGrid" x="607" y="346" w="570" h="219" measuredW="516" measuredH="219" zOrder="3" locked="false" isInGroup="-1"> <controlProperties> <align>center</align> <borderStyle>none</borderStyle> <hLines>false</hLines> <map>%3Carea%20shape%3D%22rect%22%20coords%3D%22366...</map> <rowHeight>33</rowHeight> <text>Flight%2C%20No%20Bag%2C%20Classic%2C%20Flex%...</text> <value>15</value> <verticalScrollbar>true</verticalScrollbar> <vLines>true</vLines> </controlProperties> </control></pre>	-
	When I click on “ Book ”	<pre><control controlID="34" controlTypeID="com.balsamiq.mockups::Button" x="1097" y="665" w="-1" h="-1" measuredW="51" measuredH="27" zOrder="6" locked="false" isInGroup="-1"> <controlProperties> <state>disabled</state> <text>Book</text> </controlProperties> </control></pre>	-
	Then will be displayed “ Jaccepte les Conditions d'achat concernant le(s) tarif(s) aérien(s). ” (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	-	Dynamic behavior between screens. Untraceable interaction.

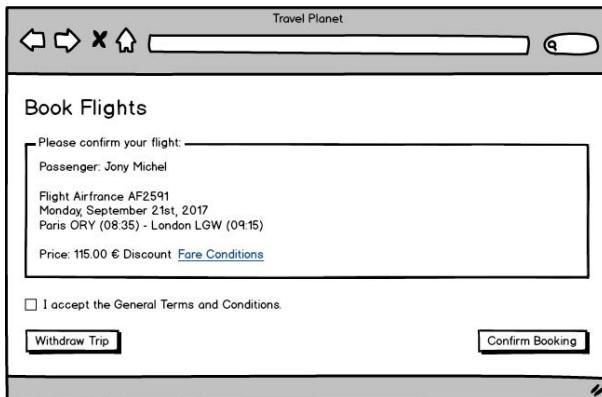


Figure 113. UI prototype for confirming a booking (first version).

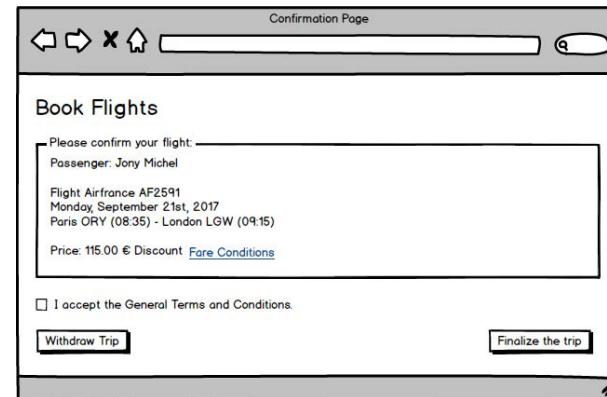


Figure 114. UI prototype for confirming a booking (revised version after testing).

Scenario: Confirm a Flight Selection (Full Version)			
Battery	Step	Balsamiq extract (XML source file)	Error
1	Given "Confirmation Page" is displayed (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	-	"Confirmation Page" does not exist.
2	Given "Confirmation Page" is displayed	<control controlID="0" controlTypeID="com.balsamiq.mockups::BrowserWindow" x="567" y="146" w="651" h="425" measuredW="450" measuredH="400" zOrder="0" locked="false" isInGroup="-1"> <controlProperties> <text>Confirmation%20Page</text> </controlProperties> </control>	-
	When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	<control controlID="30" controlTypeID="com.balsamiq.mockups::CheckBox" x="588" y="455" w="-1" h="-1" measuredW="276" measuredH="22" zOrder="5" locked="false" isInGroup="-1"> <controlProperties> <text>I%20accept%20the%20General%20Terms%20and%20Conditions.</text> </controlProperties> </control>	-
	And I click on "Finalize the trip" (FAILED) (java.lang.AssertionError: expected:<1> but was:<0>)	<control controlID="29" controlTypeID="com.balsamiq.mockups::Button" x="1074" y="493" w="-1" h="-1" measuredW="119" measuredH="27" zOrder="4" locked="false" isInGroup="-1"> <controlProperties> <text>Confirm%20Booking</text> </controlProperties> </control>	Expected field "Finalize the trip" but was "Confirm Booking".

3	<p>And I click on “Finalize the trip”</p>	<pre><control controlID="29" controlTypeID="com.balsamiq.mockups::Button" x="1074" y="493" w="-1" h="-1" measuredW="119" measuredH="27" zOrder="4" locked="false" isInGroup="-1"> <controlProperties> <text>Finalize%20the%20trip</text> </controlProperties> </control></pre>	-
	<p>Then will be displayed “Votre voyage a été confirmé!” (FAILED) (<code>java.lang.AssertionError: expected:<1> but was:<0></code>)</p>	-	<p>Dynamic behavior between screens. Untraceable interaction.</p>



Figure 115. UI prototype: Trip Confirmed.

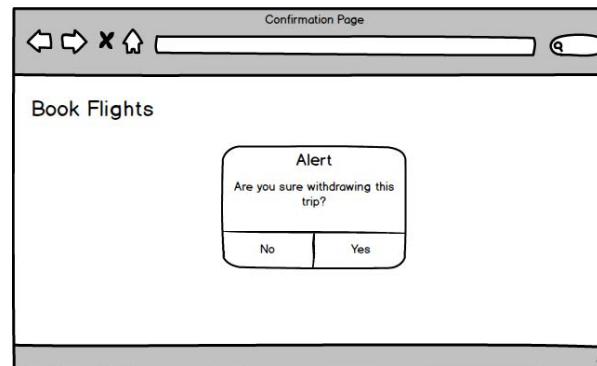


Figure 116. UI prototype: Withdrawing confirmation.

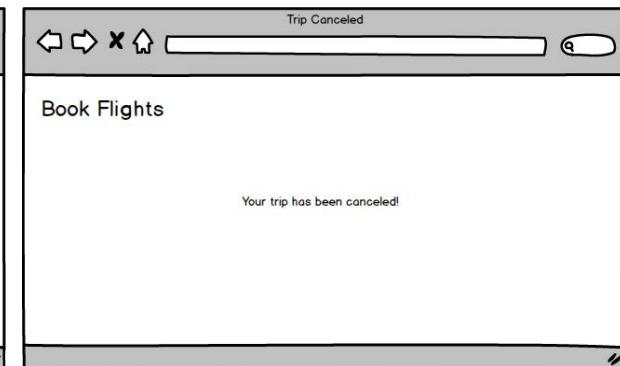


Figure 117. UI prototype: Trip Canceled.

Table 31. Test results in Balsamiq prototypes.

Figure 118. UI prototype: Multidestination search.

In the fifth battery of tests, the step 9 passed, and an error was found in the step 10 (When I set '6' in the field 'Timeframe'). The field "Timeframe" was named as "Time Frame". The field was renamed in the prototype to fix the inconsistency. The same occurred in the sixth and seventh battery of tests, respectively with the fields "Direct Flights Only" (step 11) and "Flight Class" (step 12). They were named as "Only direct flights" and "Class" respectively. They were also renamed, so the test passed.

In the eighth battery of tests, an error was found in the step 13 (And I set 'Air France' in the field 'Companies')). Three elements "SearchBox" were found to address the same field named only as "Companies". The solution was to identify uniquely each one of the fields "SearchBox", once each one of them is able to receive different values during the interaction. If we redesign the step to call specifically one of the fields (e.g. Company 1) the test passes, as we are interacting with just a unique and determined field. If otherwise we call the group Companies as a whole, we do not know with which field we should interact. The three fields were named respectively as "Company 1", "Company 2" and "Company 3", leaving the name "Companies" to reference only the group as a whole. Once again, both the scenario and the prototype have been updated.

In the ninth battery of tests, the steps 13 and 14 passed. For the step 15, at the end of the first scenario, the message referenced by the last step is supposed to be displayed in another screen as a result of the interaction. As stated in chapter 6, tests on prototypes at this level of refinement do not consider the dynamic aspect of the interaction, so tests like this, involving navigation between screens, will always fail.

Figure 119. UI prototype: Flight selected.

Following the booking process, the second scenario “Select a Return Flight Searched With Full Options” ran only 3 batteries of tests until get a consistent prototype. The first battery found an error in the element “Availability Page” that had not been found in the prototype. In the second battery, the field “No Bag” was named as “Discount” in the grid. Finally, the third battery fell in the case mentioned previously, which consists in checking a message that is supposed to be displayed in the next screen as a result of the interaction.

The third and last scenario to conclude the booking (“Confirm a Flight Selection Full Version”), also ran only 3 batteries of tests until get a consistent prototype. The first one found the same error related to the name of the page. In the second one, the button “Finalize the trip” was named as “Confirm Booking”, and the third and last battery felt in the case of dynamic behavior between screens. Notice that the message “I accept the General Terms and Conditions” in English was considered equivalent to the message “J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).” in French.

In our further test releases with other scenarios, we got errors when testing the steps “And I choose ‘One-way Trip’” and “When I choose ‘Multidestination Trip’” because these options do not exist in the UI prototypes for searching flights. In fact, the correspondent option was named “One-way / Multidestination”. Here we get an important inconsistency identified with the task model. In the test of our extracted scenarios from the task models presented in the previous section, we can notice that three tasks were modeled to select the trip type: one-way, roundtrip, or multidestination. However, in this version of the prototype, it has been modeled only two options: one for choosing a roundtrip, and another for choosing a one-way / multidestination trip. This option has been made for the prototype because, in terms of interaction, the action required for providing data for multidestination flights is actually the same of the one for providing data for a set of one-way flights. In terms of user requirements, this is a conflicting specification, so such an inconsistency must be shown up. Thus, either the prototype should follow what is specified in the task model, or the task model should be fixed to support the interaction in the way proposed by the prototype.

In the scenario “Successful Multidestination Tickets Search”, our algorithm has identified, as expected, three fields named “Departure” and “Destination” when running respectively the steps “And I inform ‘Toulouse’ and choose ‘Toulouse, Blagnac (TLS)’ in the field ‘Departure’” and “When I inform ‘Paris’ and choose ‘Paris, Charles-de-Gaulle (CDG)’ in the field ‘Destination’” [(FAILED) (java.lang.AssertionError: expected:<1> but was:<3>)]. As the designer just probably replicated (copied and pasted) the three fields with the same name, with the purpose of illustrating the change on the UI when the “One-way / Multidestination” option is selected, the group to which such fields belonged has been maintained, so this set up the inconsistency. Otherwise, if the fields had the same name, but belonged to different groups, an inconsistency would not be signalized as it would indicate that the fields were intentionally modeled as different objects.

Finally, for the following step “And I set ‘Sam, Déc 1, 2018’ in the field ‘Departure Date’” in the same scenario, the field “Departure Date” was also replicated, but the designer did not associate the pair of elements (labels and actual fields) to a group, i.e. each element (label and field) has been found belonging to distinct groups in each instance of the field “Departure Date”. The inconsistency was also detected and signalized.

8.5.1. Types of Inconsistencies Identified

By summarizing the results presented above, below we formalize the types of inconsistencies found by our testing approach when assessing the Balsamiq prototypes:

- **Conflict between expected and actual elements**, refers to elements that are specified with different names in the step and in the prototype.
- **Element and label in different groups**, refers to the absence of group links between labels and the actual interactive element in the prototype.
- **Inexistent elements**, refers to the real absence in the prototypes of elements that are specified in the step.
- **Element semantically inconsistent**, refers to the use of interaction elements in the prototype that are semantically inconsistent with the behavior they are supposed to model.
- **More than one element to represent the same field**, is caused when there are at least two elements (or more) in the prototype which are of the same type and are placed in the same group (or have the same name) of the searched field.
- **Untraceable interaction between screens**, refers to the cases where the interaction changes the state of the interface, which is not identified in prototypes with the level of requirement we are considering.

8.6. Assessing Final UIs

Unlike Balsamiq prototypes, testing on final UIs runs directly on the user interface, mimicking all the actions that would be performed by a real user. However, despite the fact that we should manually locate the identifiers of each interaction element on the interface and assign them in the “MyPage” class (as detailed in chapter 6), the process of testing runs exactly like on the UI prototypes, i.e. the algorithm parses each step of the User Stories at a time. It means that if an error is found in a given step, the test stops until it has been fixed. The testing of the final UIs in our case study was conducted directly on the UIs of the current system for booking business travel in our institute. The system is hosted in our intranet, so an additional story to access the system from our intranet login page was necessary. This story is presented below:

<p>User Story: Access to Travel Planet</p> <p>Narrative: As a UPS registered user I want to be able to reach the system feature of searching flights So that I get access to the Travel Planet system</p> <p>Scenario: Proceed to Login</p> <p>Given I go to "UPS Login Page" When I set "username" in the field "Username" And I set "password" in the field "Password" When I click on "Login" Then will be displayed "Intranet (Personnel administratif et technique-Enseignants)"</p> <p>Scenario: Reach the Travel Planet Search Page</p> <p>Given I go to "Travel Planet Search Page" When I click on "Réservations Online" And I click on "Réserver" When I click on "Avion" Then will be displayed "Avion"</p>
--

The first battery of tests has identified an error with the step “Then will be displayed ‘2. Sélectionner un voyage’” in the first scenario “Successful Roundtrip Tickets Search”. The current message displayed by the system is actually “Choisissez vos vols aller et retour, puis cliquez sur Réserver.”, so the step was updated. Following this, when running the second scenario “Successful Roundtrip Tickets Search With Full Options”, the second battery of tests identified a problem

with the identification/location of the field “Departure Time Frame”. The same occurred with the field “Arrival Time Frame” (third battery). In the fourth battery, the test identified the absence of the field “Number of Passengers” on the UI. In fact, unlike the task model and the Balsamiq prototype, the final UI did not implement this field, so it is an important inconsistency to be verified. In fifth and sixth batteries, the fields “Timeframe” and “Flight Class” were not located as well, due to the same reason of the fields “Departure Time Frame” and “Arrival Time Frame”. We noticed that these four fields are Selects (Combo Boxes), so for some unknown reason, the implementation of such fields on the final UI does not allow to identify them either using IDs or XPaths. As during this study, we had no access to the source code of the application to implement some correction and run the test again, we decided to cut the respective steps off the scenario.

In the seventh battery, the test identified an error with the length of the field “Company 1”. The Text Field implemented on the UI supports only two characters, so the value “Air France” does not fit. In fact, the user must inform a two-character internal code for the company he/she wants to select. In this case, the appropriate code for “Air France” is “AF”, so the value in the step was updated to this value. In the eighth battery of tests, all the steps for the second scenario succeeded running, and the third scenario “Successful One-way Tickets Search” started to run. An error was identified just in the last step where the message “Choisissez vos vols aller et retour, puis cliquez sur Réserver.” was expected, but the message shown on the UI was “Choisissez vos vols, puis cliquez sur Réserver.”. The step was adjusted appropriately to make the test passes.

In the ninth battery running the scenario “Successful Multidestination Tickets Search”, the step “When I inform ‘Paris’ and choose ‘Paris, Charles-de-Gaulle (CDG)’ in the field ‘Departure’” has failed once the field “Departure” had already been filled before with “Toulouse, Blagnac (TLS)” as the first departure of a multidestination trip. The field had to be renamed to correctly identify the second departure field. It was named as “Departure 2”. The same solution was applied to the second instances of “Destination” (that was renamed to “Destination 2”), and “Departure Date” (that was renamed to “Departure Date 2”).

In the tenth battery of tests, an error was identified just in the last step where the message “Choisissez vos vols, puis cliquez sur Réserver.” was expected, but the system actually showed a different message for multidestination trips. It shows “Choisissez vos vols ou trains, puis cliquez sur Réserver.”. The step was adjusted appropriately to make the test passes. Finally, the eleventh battery got all the scenarios passed and then the User Story “Flight Ticket Search” could be entirely validated.

In the twelfth battery of tests, an error was found in the step “And I click on ‘No Bag’ referring to ‘Air France 7522’” for the scenario “Select a Return Flight Searched Without Full Options”. The field “No Bag” has already been filled by the previous step “When I click on ‘No Bag’ referring to ‘Air France 7519’”, so the test fails. Besides that, the flight Air France 7522 was not available for booking anymore, so we changed for the flight Air France 7518. At the end, the solution was to give different names for each field referencing each mentioned flight. So, both steps were rewritten to “When I click on ‘Air France 7519’ referring to ‘No Bag’” and “And I click on ‘Air France 7518’ referring to ‘No Bag’” in order to create unique identifiers for the flights.

The thirteenth battery of tests run successfully the scenarios “Select a Return Flight Searched With Full Options” and “Select a One-way Flight”, but stopped with an error in the step “And I click on ‘Air France 7700’ referring to ‘No Bag’” for the scenario “Select a Multidestination Flight”. For multidestination trips, the final version of the UI actually added an additional step before reaching the second flight leg. The user must now select the first flight leg, put the flight in

a basket, and only then select the flight for the second flight leg. In terms of interaction, such a decision is inconsistent with the user requirements previously described in task models and prototypes, so the test failed, and the inconsistency is shown up. Once more, in a real case of software development, designers and requirements engineers must decide which interaction solution would be picked up, update the models accordingly, and then run new batteries of regression tests to ensure everything is consistent. For the fourteenth battery of tests, we updated the respective scenario to add this additional step. Additionally, we also changed for the flight “easyJet 3985” once the “Air France 7700” was not available anymore. That got all the scenarios passed and then the User Story “Select a Suitable Flight” could be entirely validated.

In the fifteenth and last battery of tests, we got all the remaining scenarios for the User Story “Confirm Flight Selection” passed. Nonetheless, we intentionally did not conclude the four first scenarios once they would effectively register a fake business trip for the user, so they were set as pending. Notice that the last scenario “Decline a Flight Selection” was updated both in Balsamiq prototypes and in the final UIs. A last step for confirming the withdrawal through a dialog box was added, and the agreement with the general terms and conditions was removed.

Table 32 below shows all the results of the 15 batteries of tests, highlighting step by step all the errors found, and the respective interaction elements affected by them. Screenshots of the final UIs under testing are also presented along with the scenarios (Figure 120, Figure 121, Figure 122, Figure 123, Figure 124, Figure 125 and Figure 126).

8.6.1. Types of Inconsistencies Identified

By summarizing the results presented above, below we formalize the types of inconsistencies found by our testing approach when assessing the final UIs:

- **Message not identified**, refers to messages that are changing constantly, or to the presence of conflicting messages.
- **Element or value not found**, refers to fields or values that are expected to be shown on the user interface (and are able to be identified by the locators there) but, due to the dynamic data behavior in the system, are not shown up.
- **Inexistent elements**, refers to elements that are mentioned in the step as part of the requirements specification, but simply have not been implemented on the final UI.
- **Values that do not fit the field**, refers to values mentioned in the step that do not fit the field they were designed to fill in.
- **Fields already filled in**, refers to fields that were already filled in when a given step tries to reach them.
- **Element not identified**, refers to elements that do not carry a unique and single identifier (or carry a dynamic generated one) and/or cannot be reached by using their XPaths.

Figure 120. Final UI for searching flights.

Figure 121. Final UI for searching multidestination flights.

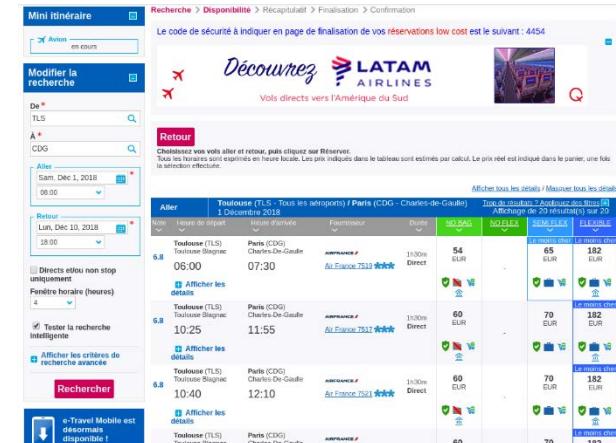


Figure 122. Final UI for choosing flights.

Battery	Scenario: Successful Roundtrip Tickets Search			
	Step	Error	Interaction Element Affected	
1	Proceed to Login	-		-
	Reach the Travel Planet Search Page	-		-
	Given I go to "Flight Search"	-		-
	When I select "Round Trip"	-		-
	And I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	-		-
	When I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	-		-
	And I set "Sam, Déc 1, 2018" in the field "Departure Date"	-		-
	When I set "Lun, Déc 10, 2018" in the field "Arrival Date"	-		-
	And I submit "Search"	-		-

	Then will be displayed “ 2. Sélectionner un voyage ” (FAILED)	Message not identified	<p class="availHint">Choisissez vos vols aller et retour, puis cliquez sur Réserver.</p>
2	Then will be displayed “Choisissez vos vols aller et retour, puis cliquez sur Réserver.”	-	-
	Scenario: Successful Roundtrip Tickets Search With Full Options		
	Reach the Travel Planet Search Page	-	-
	Given I go to “Flight Search”	-	-
	When I inform “Toulouse” and choose “Toulouse, Blagnac (TLS)” in the field “Departure”	-	-
	And I inform “Paris” and choose “Paris, Charles-de-Gaulle (CDG)” in the field “Destination”	-	-
	When I set “Sam, Déc 1, 2018” in the field “Departure Date”	-	-
3	And I set “08:00” in the field “ Departure Time Frame ” (FAILED)	Element not identified	@ElementMap(name = "Departure Time Frame", locatorType = ElementLocatorType.XPath, locator = "/@*[@id='tripDate_1']/fieldset/div/span/div/select") private Select DepartureTime;
	And I set “08:00” in the field “Departure Time Frame”	-	-
	When I choose “Round Trip”	-	-
	And I set “Lun, Déc 10, 2018” in the field “Arrival Date”	-	-
4	When I set “10:00” in the field “ Arrival Time Frame ” (FAILED)	Element not identified	@ElementMap(name = "Arrival Time Frame", locatorType = ElementLocatorType.XPath, locator = "/@*[@id='tripDate_2']/fieldset/div/span/div/select") private Select ReturnTime;
	When I set “10:00” in the field “Arrival Time Frame”	-	-
5	And I choose the option of value “2” in the field “ Number of Passengers ” (FAILED)	Element not found in “Flight Search”	-
	And I choose the option of value “2” in the field “Number of Passengers”	-	-
6	When I set “6” in the field “ Timeframe ” (FAILED)	Element not identified	@ElementMap(name = "Timeframe", locatorType = ElementLocatorType.XPath, locator = ".//*[@id='atwsel29'])" private Select TimeFrame;
	When I set “6” in the field “Timeframe”	-	-
	And I select “Direct Flights Only”	-	-
	When I choose the option of value “Economique” in the field “ Flight Class ” (FAILED)	Element not identified	@ElementMap(name = "Flight Class", locatorType = ElementLocatorType.XPath, locator = ".//*[@id='acsse124'])" private Select FlightClass;

7	When I choose the option of value “Economique” in the field “Flight Class”	-	-
	And I set “Air France” in the field “Company 1” (FAILED)	Value does not fit the field	@ElementMap(name = "Company 1", locatorType = ElementLocatorType.XPath, locator = "//*[@id='Fp1']") private TextField CompanyOne;
	And I set “AF” in the field “Company 1”	-	-
	When I submit “Search”	-	-
	Then will be displayed “Choisissez vos vols aller et retour, puis cliquez sur Réserver.”	-	-
Scenario: Successful One-way Tickets Search			
8	Reach the Travel Planet Search Page	-	-
	Given I go to “Flight Search”	-	-
	When I inform “Toulouse” and choose “Toulouse, Blagnac (TLS)” in the field “Departure”	-	-
	And I inform “Paris” and choose “Paris, Charles-de-Gaulle (CDG)” in the field “Destination”	-	-
	When I set “Sam, Déc 1, 2018” in the field “Departure Date”	-	-
	And I choose “One-way / Multidestination”	-	-
	When I submit “Search”	-	-
	Then will be displayed “Choisissez vos vols aller et retour, puis cliquez sur Réserver.” (FAILED)	Message not identified	<p class="availHint">Choisissez vos vols, puis cliquez sur Réserver.</p>
	Then will be displayed “Choisissez vos vols, puis cliquez sur Réserver.”	-	-
Scenario: Successful Multidestination Tickets Search			
9	Reach the Travel Planet Search Page	-	-
	Given I go to “Flight Search”	-	-
	When I choose “One-way / Multidestination”	-	-
	When I inform “Toulouse” and choose “Toulouse, Blagnac (TLS)” in the field “Departure”	-	-
	And I inform “Paris” and choose “Paris, Charles-de-Gaulle (CDG)” in the field “Destination”	-	-
	When I set “Sam, Déc 1, 2018” in the field “Departure Date”	-	-
	When I inform “Paris” and choose “Paris, Charles-de-Gaulle (CDG)” in the field “Departure” (FAILED)	“Departure” already filled	@ElementMap(name = "Departure", locatorType = ElementLocatorType.XPath, locator = ".//*[@id='B_LOCATION_1']","/html/body/div[1]/div[1]/table/tbody/tr/td[2]/div[5]/div[2]/div[2]/fo

			<pre>rm/div/div[2]/div/div[3]/div[1]/div/div[1]/div[1] /div/div/div[2]/div"}) private AutoComplete From;</pre>
10	When I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Departure 2"	-	-
	And I inform "Nice" and choose "Nice, Côte D'Azur (NCE)" in the field "Destination 2"	-	-
	When I set "Lun, Déc 10, 2018" in the field "Departure Date 2"	-	-
	And I submit "Search"	-	-
	Then will be displayed "Choisissez vos vols, puis cliquez sur Réserver." (FAILED)	Message not identified	<p class="availHint">Choisissez vos vols ou trains, puis cliquez sur Réserver.</p>
11	Then will be displayed "Choisissez vos vols ou trains, puis cliquez sur Réserver."	-	-
	Scenario: Search for Flights More Than One Year in Advance		
	Reach the Travel Planet Search Page	-	-
	Given I go to "Flight Search"	-	-
	When I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	-	-
	And I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	-	-
	When I set "Dim, Déc 1, 2019" in the field "Departure Date"	-	-
	When I choose "One-way / Multidestination"	-	-
	And I submit "Search"	-	-
	Then will be displayed "Erreur : Vous devez choisir une date de départ ultérieure comprise entre 4 heures et 11 mois. Veuillez sélectionner une autre date. (10032)"	-	-
Scenario: Search for a Return Flight Before a Departure Flight			
11	Reach the Travel Planet Search Page	-	-
	Given I go to "Flight Search"	-	-
	When I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "Departure"	-	-
	And I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"	-	-
	When I set "Lun, Déc 10, 2018" in the field "Departure Date"	-	-
	And I choose "Round Trip"	-	-
	When I set "Sam, Déc 1, 2018" in the field "Arrival Date"	-	-

	And I submit “Search”	-	-
	Then will be displayed “Erreur : La date de retour ne peut pas être antérieure à la date de départ.”	-	-

The screenshot shows a flight search interface with two flight options listed:

- Flight 1:** Toulouse (Toulouse Blagnac) to Paris (Charles De Gaulle) on Air France 7519 at 06:00 on 1 December 2018, returning at 07:30 on 1 December 2018. The total price is 130.01 EUR.
- Flight 2:** Paris (Charles De Gaulle) to Toulouse (Toulouse Blagnac) on Air France 7522 at 14:55 on 10 December 2018, returning at 16:15 on 10 December 2018. The total price is 130.01 EUR.

Both flights are labeled as "Economique avec restrictions". A "Réservé" button is visible for both flights.

Figure 123. Final UI with the selected flights.

The screenshot shows a confirmation page for the selected flights:

- Flight 1:** Toulouse (Toulouse Blagnac) to Paris (Charles De Gaulle) on Air France 7519 at 06:00 on 1 December 2018, returning at 07:30 on 1 December 2018. Total price: 130.01 EUR.
- Flight 2:** Paris (Charles De Gaulle) to Toulouse (Toulouse Blagnac) on Air France 7522 at 14:55 on 10 December 2018, returning at 16:15 on 10 December 2018. Total price: 130.01 EUR.

The page also displays a note: "Vé le moins cher : Tarif saison le plus bas proposé dans la classe 77 00 EUR sélectionnée."

Figure 124. Final UI for confirming the selected flights.

12	Scenario: Select a Return Flight Searched Without Full Options	-	-
	Successful Roundtrip Tickets Search	-	-
	Given “Availability Page” is displayed	-	-
	When I click on “No Bag” referring to “Air France 7519”	-	-
	And I click on “No Bag” referring to “Air France 7522” (FAILED)	“No Bag” already filled	@ElementMap(name = "No Bag", locatorType = ElementLocatorType.XPATH, locator = "//*[@id='w1_0_c0_r1'])") private Button NoBag;
13	When I click on “Air France 7519” referring to “No Bag”	-	-

	And I click on “Air France 7518” referring to “No Bag”	-	-
	When I click on “Book”	-	-
	Then will be displayed “J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).”	-	-
Scenario: Select a Return Flight Searched With Full Options			
	Successful Roundtrip Tickets Search With Full Options	-	-
	Given “Availability Page” is displayed	-	-
	When I click on “Air France 7519” referring to “No Bag”	-	-
	And I click on “Air France 7522” referring to “No Bag”	-	-
	When I click on “Book”	-	-
	Then will be displayed “J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).”	-	-
Scenario: Select a One-way Flight			
	Successful One-way Tickets Search	-	-
	Given “Availability Page” is displayed	-	-
	When I click on “Air France 7519” referring to “No Bag”	-	-
	And I click on “Book”	-	-
	Then will be displayed “J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).”	-	-
Scenario: Select a Multidestination Flight			
	Successful Multidestination Tickets Search	-	-
	Given “Availability Page” is displayed	-	-
	When I click on “Air France 7519” referring to “No Bag”	-	-
	And I click on “ Air France 7700 ” referring to “No Bag” (FAILED)	Element “Air France 7700” not found @ElementMap(name = "Air France 7700", locatorType = ElementLocatorType.XPATH, locator = "//*[@id='w1_0_c0_r22']") private Button AirFrance7700;	
14	And I click on “Book”	-	-
	When I click on “easyJet 3985” referring to “No Bag”	-	-
	And I click on “Book”	-	-

	Then will be displayed “J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s).”	-	-
--	--	---	---

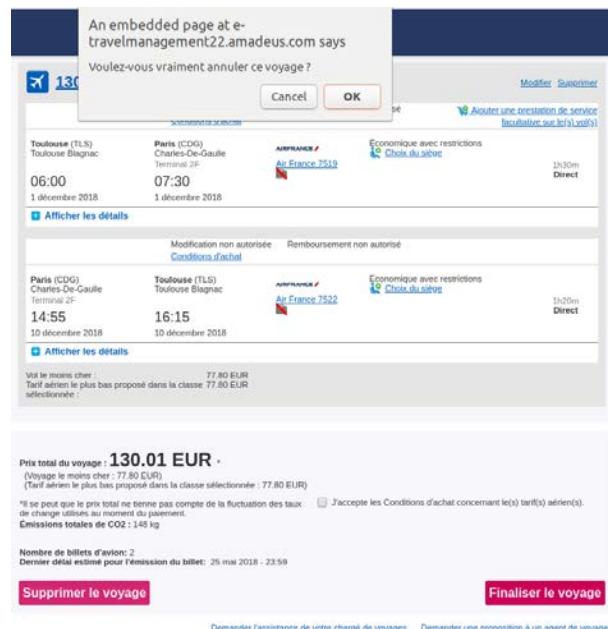


Figure 125. Final UI: dialog box before canceling.



Figure 126. Final UI: trip canceled.

Scenario: Confirm a Flight Selection			
15	Select a Return Flight Searched Without Full Options	-	-
	Given "Confirmation Page" is displayed	-	-
	When I choose "I accept the General Terms and Conditions."	-	-
	And I click on "Finalize the trip" (NOT PERFORMED)	-	-
	Then will be displayed "Votre voyage a été confirmé!" (NOT PERFORMED)	-	-
Scenario: Confirm a Flight Selection (Full Version)			
	Select a Return Flight Searched With Full Options	-	-

Given “Confirmation Page” is displayed	-	-
When I choose “I accept the General Terms and Conditions.”	-	-
And I click on “Finalize the trip” (NOT PERFORMED)	-	-
Then will be displayed “Votre voyage a été confirmé!” (NOT PERFORMED)	-	-
Scenario: Confirm a Flight Selection for a One-Way Trip		
Select a One-way Flight	-	-
Given “Confirmation Page” is displayed	-	-
When I choose “I accept the General Terms and Conditions.”	-	-
And I click on “Finalize the trip” (NOT PERFORMED)	-	-
Then will be displayed “Votre voyage a été confirmé!” (NOT PERFORMED)	-	-
Scenario: Confirm a Flight Selection for a Multidestination Trip		
Select a Multidestination Flight	-	-
Given “Confirmation Page” is displayed	-	-
When I choose “I accept the General Terms and Conditions.”	-	-
And I click on “Finalize the trip” (NOT PERFORMED)	-	-
Then will be displayed “Votre voyage a été confirmé!” (NOT PERFORMED)	-	-
Scenario: Decline a Flight Selection		
Select a One-way Flight	-	-
Given “Confirmation Page” is displayed	-	-
When I click on “Decline the trip”	-	-
Then will be displayed “Voulez-vous vraiment annuler ce voyage ?” in the dialog box	-	-
When I confirm the dialog box	-	-
Then will be displayed “Votre voyage a été annulé.”	-	-

Table 32. Test results on the final UI.

8.7. Results Mapping

In the last three sections, we have presented the results of tests conducted individually in each one of the three target artifacts we selected: task models, UI prototypes and the final user interfaces. By mapping such results and putting them together, we can build a complete traceability overview of the steps in User Stories and identify how inconsistent they were modeled in the different artifacts.

Table 33 brings, for each considered artifact, the mapping of results of the first battery of tests in each step of the full scenario for booking a roundtrip. As some steps were being updated after having previously failed in a given artifact, the results shown in yellow in the table below indicate that, for the artifact in question, the test run with an updated version of the step and still failed. Results shown in blue indicate that, for the artifact in question, the test run with an updated version of the step and the test passed with such a version. Results shown in green indicate steps that passed the test in that artifact, and results in red indicate steps that failed in that artifact. Finally, results shown in orange indicate that such a step has been pending in that artifact, it is the case of the steps that effectively conclude the booking on the final UI. We avoided such steps to do not create fake reservations in the booking system of your institute. In the column User Stories/Scenarios, we considered the original steps, as conceived before starting the first battery of tests in any artifact. Notice that once some step of scenario for some artifact fails, the scenario is considered as failed as well.

Analyzing the results of mapping presented above for the first scenario “Successful Roundtrip Tickets Search With Full Options”, we notice that the first step (that has succeeded in the task model) failed when tested with the Balsamiq prototypes. The reason is that the prototype had not addressed the web pages correctly, i.e. the “Book Flights” page could not be identified there. In a following battery of tests, this page has ended up being named “Flight Search” instead, which made the test passes when running on the Final UI.

The two following steps have failed for task models but passed for Balsamiq prototypes and Final UI. As analyzed in section 8.4, the reason of failure in task models is due to the additional tasks “Provide List of Airports” for the group of tasks which provides information of departure and destination in the task model, from the second step until the eighth step, the gap between the expected position and the actual one was exactly two positions. However, both steps passed when tested for the Balsamiq prototype and the Final UIs, once the UI element was correctly represented, i.e. as a “SearchBox” in the prototype, and as an “Autocomplete” field in the Final UI. The step testing the field “Departure Date” nonetheless failed for the Balsamiq prototype but passed for the Final UI (the same has occurred with the field “Arrival Date”). The reason of failure in the prototype is that the label of the field and the UI element itself were not represented in the same group of elements. Contrasting with that, the step testing the field “Departure Time Frame” passed for the Balsamiq prototype but failed for the Final UI (the same has occurred with the field “Arrival Time Frame”). The reason is that was not possible to locate unique identifiers for the element on the Final UI.

At the sixth step (“When I choose ‘Round Trip’”), as the task for choosing the “Round Trip” has not been exported from the task model to the scenario, the gap from the eighth position in task model scenarios until the end of the scenario (excluding the tasks not found) dropped down from two to one position. Such step succeeded when identifying the element “Round Trip” in the Balsamiq prototype and in the Final UI. The element “Number of Passengers” at the ninth step was not found both in the Balsamiq prototype and in the Final UI, despite being specified

in the task model scenario (although not in the right position). The element “Timeframe” at the tenth step was written as “Time Frame” in the Balsamiq prototype and was misspecified with an unknown behavior “Adjust Timeframe” in the task model, so it could not be identified in these artifacts. The same has occurred with the element “Flight Class” at the twelfth step which was written as just “Class” in the Balsamiq prototype and specified as “Define Fight Class” in the task model, an unknown behavior. In the Final UI, they have not been identified as well, but due to the problem of unique identifiers.

The element “Direct Flights Only” at the following step was written as “Only direct flights” in the Balsamiq prototype, so was not identified, but was correctly written in the Final UI and was rightly identified there. The correspondent task for this step was in the wrong position in the task model scenario. The element “Companies” was misspecified with an unknown behavior “Define Companies” in the task model. In the Balsamiq prototype, it was addressing three different “SearchBoxes”, so it could not be identified as a unique and single element. After both the correspondent step and the prototype are redesigned to identify each field separately, the step should pass the test in the Final UI but was failed as well because the value informed on it (“Air France”) did not fit the correspondent Text Field which only accepted 3 characters. In this case, the step was fixed to inform only the value “AF”, the correspondent code defined to be used in the Final UI.

The button “Search” was correctly identified both in the Balsamiq prototype and the Final UI, but the referenced task in the scenario extracted from the task model was found in the wrong position. Finally, for the first scenario, the message resultant from the user interaction when searching flights was not identified in the scenario tasks and was not reachable in the Balsamiq prototype due to the untraceable interaction between screens. In the Final UI, the message to check was modified, so the step was refactored to reference the new message. Thereby, after the modification, the test passed for this artifact.

For the steps in the second and third scenarios, all of them failed for the task model and a deep work for fixing the compatibility issues would be required. Specification of tasks did not follow the behaviors mapped in the ontology, so none of them could be identified during the test. The prototyping of web pages that should be displayed when starting those scenarios failed once they addressed wrong page names. They were correct in the final UIs nonetheless. The choice of flights in the second and third steps of the second scenario failed in the Balsamiq prototype (the name of the element was misidentified) and were refactored to requiring an action of clicking on the number of the flights (instead of on the fare profile), so they passed the test in the Final UI. The behavior of clicking on the button “Book” was correctly addressed in both the Balsamiq prototype and in the final UI. The checking of message after the interaction was succeeded in the final UI but failed as expected in the Balsamiq prototype due to the untraceable interaction between screens.

User Stories	Scenario from Task Model	Balsamiq Prototype	Final UI
User Story: Flight Tickets Search Narrative: As a IRIT researcher I want to be able to search air tickets for my business trips, providing destinations and dates So that I can obtain information about rates and times of the flights.	-	-	-
Scenario: Successful Roundtrip Tickets Search With Full Options	FAILED	FAILED	FAILED
Given I go to “Book Flights”	Expected: 1 Actual: 1	Expected: 1 Actual: 0	Expected: Flight Search Actual: Flight Search
When I inform “Toulouse” and choose “Toulouse, Blagnac (TLS)” in the field “Departure”	Expected: 2/3 Actual: 0	Expected: 1 Actual: 1	Expected: Departure Actual: Departure
And I inform “Paris” and choose “Paris, Charles-de-Gaulle (CDG)” in the field “Destination”	Expected: 4/5 Actual: 0	Expected: 1 Actual: 1	Expected: Destination Actual: Destination
When I set “Sam, Déc 1, 2018” in the field “Departure Date”	Expected: 6 Actual: 8	Expected: 1 Actual: 0	Expected: Departure Date Actual: Departure Date
And I set “08:00” in the field “Departure Time Frame”	Expected: 7 Actual: 9	Expected: 1 Actual: 1	Expected: Departure Time Frame Actual: Element not identified
When I choose “Round Trip”	Expected: 8 Actual: 0	Expected: 1 Actual: 1	Expected: Round Trip Actual: Round Trip
And I set “Lun, Déc 10, 2018” in the field “Arrival Date”	Expected: 9 Actual: 10	Expected: 1 Actual: 0	Expected: Arrival Date Actual: Arrival Date
When I set “10:00” in the field “Arrival Time Frame”	Expected: 10 Actual: 11	Expected: 1 Actual: 1	Expected: Arrival Time Frame Actual: Element not identified
And I choose the option of value “2” in the field “Number of Passengers”	Expected: 11 Actual: 12	Expected: 1 Actual: 0	Expected: Number of Passengers Actual: Element not found in “Flight Search”
When I set “6” in the field “Timeframe”	Expected: 12 Actual: 0	Expected: 1 Actual: 0	Expected: Timeframe Actual: Element not identified
And I select “Direct Flights Only”	Expected: 13 Actual: 14	Expected: 1 Actual: 0	Expected: Direct Flights Only Actual: Direct Flights Only
When I choose the option of value “Economique” in the field “Flight Class”	Expected: 14	Expected: 1	Expected: Flight Class

	Actual: 0	Actual: 0	Actual: Element not identified
And I set "Air France" in the field "Companies"	Expected: 15	Expected: 1	Expected: Company 1
	Actual: 0	Actual: 0	Actual: Value does not fit the field
When I submit "Search"	Expected: 16	Expected: 1	Expected: Search
	Actual: 17	Actual: 1	Actual: Search
Then will be displayed "2. Sélectionner un voyage"	Expected: 17	Expected: 1	Expected: Proper Message
	Actual: 0	Actual: 0	Actual: Proper Message

User Story: Select a Suitable Flight Narrative: As a IRIT researcher I want to get a list of compatible flights (including their rates and times) in accordance with my search criteria So that I can select a suitable flight based on my needs.	-	-	-
Scenario: Select a Return Flight Searched With Full Options	FAILED	FAILED	PASSED
Given "Availability Page" is displayed	Expected: 18	Expected: 1	Expected: Availability Page
	Actual: 0	Actual: 0	Actual: Availability Page
When I click on "No Bag" referring to "Air France 7519"	Expected: 19	Expected: 1	Expected: Air France 7519
	Actual: 0	Actual: 0	Actual: Air France 7519
And I click on "No Bag" referring to "Air France 7522"	Expected: 20	Expected: 1	Expected: Air France 7522
	Actual: 0	Actual: 0	Actual: Air France 7522
When I click on "Book"	Expected: 21	Expected: 1	Expected: Book
	Actual: 0	Actual: 1	Actual: Book
Then will be displayed "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."	Expected: 22	Expected: 1	Expected: Proper Message
	Actual: 0	Actual: 0	Actual: Proper Message

User Story: Confirm Flight Selection Narrative: As a IRIT researcher I want to get all the required data to confirm my flights So that I can check the information, the fare rules and then finalize my booking.	-	-	-
Scenario: Confirm a Flight Selection (Full Version)	FAILED	FAILED	PENDING
Given "Confirmation Page" is displayed	Expected: 23	Expected: 1	Expected: Confirmation Page
	Actual: 0	Actual: 0	Actual: Confirmation Page
When I choose "I accept the General Terms and Conditions."	Expected: 24	Expected: 1	Expected: Proper Field
	Actual: 0	Actual: 1	Actual: Proper Field
And I click on "Finalize the trip"	Expected: 25	Expected: 1	Expected: Finalize the trip

	Actual: 0	Actual: 0	Actual: Finalize the trip
Then will be displayed “Votre voyage a été confirmé!”	Expected: 26	Expected: 1	Expected: Proper Message
	Actual: 0	Actual: 0	Actual: Proper Message

Table 33. Mapping of the results after testing.

In the third and last scenario, the behavior of accepting the general terms and conditions to confirm the booking of flights was correctly addressed in both the Balsamiq prototype and the final UI. The button “Finalize the trip” was not identified in the Balsamiq prototypes (it was “Confirm Booking”). As already explained, the action of clicking the button and verifying the confirmation message in the final UI was not performed in order to not effectively place a fake booking in the system. Due to that, both steps were signalized as “pending” in the final UI. Once more, the checking of message after the interaction failed as expected in the Balsamiq prototype due to the untraceable interaction between screens. Lastly, regarding to the final testing results in each artifact, we notice that only the second scenario was successfully executed in the final UI. The other two scenarios have failed in the other artifacts or got pending (the last scenario in the final UI).

8.8. Summary of Main Findings in the Case Study

Looking back at the types of inconsistencies we managed to identify for each artifact, we present below a summarized table (Table 34) with such types enlisted and discuss thereafter the impact of such inconsistencies when assessing the artifacts.

Task Models	Balsamiq Prototypes	Final UIs
<ul style="list-style-type: none"> • Task with different names • Task not extracted to the scenario • Different number of sequences of tasks in the task model • Wrong position • Conflict between specification and modeling • Different specification strategies • Unpaired behaviors • Equivalent behaviors missing 	<ul style="list-style-type: none"> • Conflict between expected and actual elements • Element and label in different groups • Inexistent elements • Element semantically inconsistent • More than one element to represent the same field • Untraceable interaction between screens 	<ul style="list-style-type: none"> • Message not identified • Element or value not found • Inexistent elements • Values that do not fit the field • Fields already filled in • Element not identified

Table 34. Main kinds of problems identified in each artifact after testing.

For task models, we succeeded identifying 8 different types of inconsistencies in the tested scenarios. The most common ones were the “different number of sequences of tasks in the task model”, “unpaired behaviors”, and “equivalent behaviors missing”. The first type occurs when there are more tasks in the task model scenario than steps in the User Story scenario to accomplish the same behavior. In the example presented in section 8.4, to inform a departure (or a destination) there was a sequence of 3 tasks in the scenario extracted from the task model, while in the step of the User Story scenario, a double action of informing and choosing was enough. For the second type, “unpaired behaviors” refers to tasks that would find a correspondence with the steps in the User Stories, but as they actually specify different behaviors (e.g. “Define <something>” instead of “Select <something>”), they cannot be recognized as such. “Equivalent behaviors missing” refers to behaviors that are really missing in the extracted task

model scenario, like steps that are present in the User Story, but cannot find correspondent tasks in the task model.

“Different specification strategies” comes next as the type of inconsistency incurred from specification of behaviors that could eventually aim at the same purpose, but were specified using different strategies, i.e. requiring to perform (or verify) different actions. An example from this case study is the situation in which a step of User Story scenario had described a behavior in which the system showed a message introducing a list of available flights, and the task model, a behavior in which the system provided the aforementioned list. Even with the resultant state of the system being the same in this case, the specified behaviors could not be considered equivalents once they use different specification strategies.

Tasks in “wrong positions” comes next being the type of error related to tasks that are found in different positions than their equivalent steps in User Stories. As scenarios in different conceptions are being compared when testing User Stories and task models, we consider that errors found in the sequence of tasks in the task models (compared with User Stories) are generally the most sensitive type of error, once it impacts in all other tasks in the sequence. A simple change of task positions in the beginning of a scenario invalidates the whole scenario because all the tasks in the sequence would be in wrong positions. A correction to a simple error like this would include finding the root of the problem, redesign either the step (that would impact the consistency in other artifacts) or the task model (that would imply in extracting new scenarios for testing) and run a complete battery of regression tests again. Considering that there are no other types of inconsistencies in the model, by fixing this issue (either by updating the User Story scenario to comply with the scenario extracted from the task model or updating the task model to comply with the sequence of steps from the User Story), both scenarios would become fully consistent.

“Conflicts between specification and modeling” refers to tasks modeled in the task model (and consequently exported to its scenarios) that are not present the requirements specification in the User Stories. The contrary can occur as well. This kind of inconsistency generally puts in evidence important conflicts between what is specified in the user requirements and what is effectively modeled in the artifacts. “Tasks with different names” and “Tasks not extracted to the scenario” complete the list of type of errors encountered during the tests. The first one refers to tasks that are present both in the task model and in the User Story scenario but written with a different name. The second one refers to tasks that are effectively modeled in the task model but, due to the type of operators used or the presence (or not) of other refined tasks after them in the model, causes that, during the extraction process, such tasks do not be taken to the extracted scenarios.

Concerning the type of inconsistencies registered during the test of Balsamiq prototypes, we succeeded identifying 6 different types in the tested scenarios. “Conflict between expected and actual elements” was the most frequent type and refers to elements that are specified with different names in the step and in the prototype. “Inexistent elements” and “untraceable interaction between screens” comes next and refer respectively to the real absence in the prototypes of elements that are specified in the step, and to the cases where the interaction changes the state of the interface (e.g. transitioning between screens or making appear a given value in a field). As such cases are not identified in the prototype with the level of requirement we are considering, an inconsistency is shown up.

“Elements and labels in different groups” is the next type in line and refers to one of the mechanisms of modeling used by Balsamiq. When a given UI element is composed by a label name and the interaction element itself, this encompassed structure is modeled by an entity

named “group”. Thus, to be considered as a unique and single element, both the label and the interaction element itself must be placed at the same group. If it is not the case, we are not able to reach the element and then an inconsistency is detected.

“More than one element to represent the same field” is a type of inconsistency caused when there are at least two elements (or more) in the prototype that are of the same type and are placed in the same group (or have the same name) of the searched field. Finally, the type of inconsistency named “elements semantically inconsistent” refers to the core problem we address with the ontology, i.e. the use of interaction elements in the prototype that are semantically inconsistent with the behavior they are supposed to model. This kind of inconsistency is detected when we get the list of supported interactive elements from the ontology and check if the interactive element used in the prototype is equivalent to one of them.

Concerning the type of inconsistencies registered during the test of Final UIs, we also succeeded identifying 6 different types in the tested scenarios. “Elements not identified” was the most frequent type and refers to elements that do not carry a unique and single identifier (or carry a dynamic generated one) and/or cannot be reached by using their XPaths. When observing the unsuccessful tries to find the fields “Departure Time Frame” and “Arrival Time Frame”, for example, we remarked that is a recurrent problem when automating testing on user interfaces. Some web frameworks for developing the presentation layer dynamically generates different identifiers each time the UI is charged, which makes very hard the work of previously identifying them to implement the test. Besides that, some developers skip informing unique identifiers for the fields. XPath identifiers help in most cases, but there still are some situations where the identification of locators gets very compromised.

Constant changing, or conflicting messages is another frequent issue (type of inconsistency “message not identified”). Messages sometimes change in the Final UI and the requirements specification is not updated accordingly. As a consequence, the message specified in the step to be verified in the user interface is not found on the screen. Not identifying elements or values due to dynamic data behavior is also an issue. The type of inconsistency “element or value not found” refers to fields or values that are expected to be shown on the user interface (and are able to be identified by the locators there) but, due the dynamic data behavior in the system, are not shown up. An example from the case study is a flight, which was mentioned to be verified as an example of data value in the step and was not identified in the list of resultant flights because it was not available for booking anymore. There is also the case of elements that are really nonexistent on the user interface (type of inconsistency “inexistent elements”). These elements are mentioned in the step as part of the requirements specification, but simply have not been implemented on the final UI.

Fields that were already filled in when a given step tries to reach them are also a source of inconsistencies (type of inconsistency “fields already filled in”). As happened in the case study when testing the fields “Departure” and “Destination” for a multidestination trip, due to the second flight leg, the elements were referenced with the same name more than once. When the test tried to fill in the same field a second time for the second flight leg, the inconsistency has shown up. In this case, both the step and the mapping of interaction elements on the user interface must be updated to reference unequivocally different elements for each desired interaction.

The last type of inconsistency identified refers to values mentioned in the step that do not fit the field they were designed to fill in (type of inconsistency “values that do not fit the field”). During the case study, the field “Company 1” was expected to receive the value “Air France” as

described in the step, but the concrete field “Company 1” on the user interface had been modeled to support only 3 characters. This type of inconsistency can also be extended to other incompatibilities between the type of data expected and what the field actually supports. Examples include strings to be filled in number-only fields, unformatted numbers to be filled in date-time fields, and so on.

8.9. Threats to Validity

Generalization of results. We have conducted this second study by modeling and assessing a system for booking airline tickets for business trips. Such kind of system has usually a strong search-based feature, once they are centered in providing and comparing rates, times and availability of flights given a set of provided parameters. However, as the ontology in which we based our analyses is designed for domain-independent interactive behaviors, we assume our results concerning the usefulness of our interactive behaviors would be reproduced in other interactive systems domains. Concerning the types of inconsistencies identified, we understand the list presented in this chapter is just an initial set of inconsistencies that our approach is able to identify. Further studies, especially with systems implementing different features, might reveal a broader set of inconsistencies able to be identified.

Manual reverse engineering. This study performed a manual reverse engineering of the current system in production to obtain its respective models for testing. The goal of the study was to investigate which kind of inconsistencies our approach would be able to identify in the models and in the system. Therefore, as a manual process, it was expected that inconsistencies would be naturally introduced during the modeling. Indeed, these inconsistencies were identified and that allowed us to evaluate our approach. Nonetheless, if an automated approach of reverse engineering had been used instead, such inconsistencies would probably not have taken place. Future studies should confirm this hypothesis.

Possible modeling bias. Both the conduction of the study and the interpretation and analysis of the results have been made by us. So, it is possible there has been a bias in the interpretation of such results and/or in the modeling process. To mitigate such a threat, we plan future studies considering third-part modeling and cross-checking of results by an independent reviewer in an attempt to reduce such a bias.

8.10. Conclusion and Lessons Learned

By analyzing the variety of inconsistency problems that have been identified in this case study, we can remark that some types of inconsistencies have shown to be more critical than others. While simple inconsistencies like differences in names of tasks and fields are easy to be solved, some other inconsistencies can reveal crucial problems of modeling or important incompatibilities between the requirements specification and its modeling in the artifacts. Conflicts between expected and actual elements in prototypes (usually due to different names), or messages and elements not found (or even nonexistent) in prototypes or final UIs are other examples of inconsistencies that are easy to solve.

Conflicts between specification and modeling along with different specification strategies for task models compose a more critical group of problems and must be prioritized. Concerning user interface prototypes in different levels of refinement, the presence of semantically inconsistent elements and the presence of more than one element to represent the same field are also critical groups of problems. On final UIs, fields already filled-in denotes inconsistencies that exposes important design errors.

Unpaired behaviors and equivalent behaviors missing on task models are inconsistencies directly related to the wideness of the referenced ontology. The ontology we present here (and which has been used in the case study) is always evolving and can eventually in the future support new behaviors, which would increase its capacity of recognizing other task descriptions. Tasks not extracted to the scenario, different number of task sequences in the task model, and the presence of tasks in wrong positions are problems that can have the same origin in scenarios extracted from task models, i.e. due to tasks not extracted to scenarios, those ones that have been extracted may be placed in wrong positions which will affect how many sequences of tasks the scenario lists. So, by fixing the root cause, the designer can avoid three kinds of different problems.

Element and label in different groups is an inconsistency related to the way Balsamiq implements its prototypes. It leads to a misidentification of elements but could eventually not be an issue in other prototyping tools. Likewise, elements not identified in the final UI is a specific problem of web interfaces, where not even robust XPaths can be able to identify some elements on the screen. Values that do not fit the field is another exclusive problem of final UIs, once it emerges from real data handling along the interaction.

Untraceable interaction between screens is a particular type of inconsistency due to the level of refinement we are considering for prototypes. Balsamiq is a prototype tool that actually supports a basic dialog description, allowing building links between prototypes and simulating a real navigation on user interfaces. However, we have chosen to not consider such a feature once the ontology we propose can already support more robust interactions in other levels of prototype refinements, such as the one that has been implemented by PANDA.

We notice finally that our approach makes task models, user interface prototypes and final UIs intrinsically related in such a way that a modification in the User Stories scenarios requires a full battery of regression tests in the other artifacts. For example, after having fixed the Balsamiq prototype to name the three-company field independently, we had to modify the step “And I set ‘Air France’ in the field ‘Companies’” to reference only one of them. We modified the step to “And I set ‘AF’ in the field ‘Company 1’”. When doing this, we should retest all our task model scenarios in order to ensure we did not introduce any inconsistency there due to such a modification. Maybe the task model has to be modified as well, in order to comply with the modification in the prototype. The modification has also the potential to identify new inconsistencies in the task model, that had not been identified before with the older scenario. Thus, it is imperative to work with the perspective of regression tests in mind, which reinforce the crucial importance of automated testing.

As noticed, this case study helped us to identify a wide group of inconsistency problems that can be shown up by our approach. With little effort for specifying high-level scenarios in natural language only once, we succeeded running several automated tests on the target artifacts in seconds. These results open a great range of opportunities for assessing multiple artifacts and for keeping them consistent throughout the whole software development process. Some features like the use of data providers to assess final UIs could not be tested in this case study though. This resource presented previously allows modeling steps only with data domains and injecting data on them at runtime. As we have not specified steps using this feature, we could not get results about its effectiveness in a broader case study than that one presented in chapter 6. This is indeed a good opportunity for future works.

Part IV - Conclusion

Chapter 9

Conclusion

Summary

This chapter presents the final remarks about this thesis' work. We recapitulate our achievements and discuss the main contributions and limitations of the approach. We also provide some directions for future research in this field as well as our future works already planned to be conducted for improving the proposed approach. The chapter ends with the full list of publications resultant from this thesis.

Model-based and iterative approaches are a suitable alternative to cope with the complexity of the development process of interactive systems. For that, models fulfill three main roles in the development process: decompose problems in specialized views, specify the intentions reducing ambiguity, and promote communication among stakeholders. Multiple artifacts provide specialized views for concepts handled by models, thus ensuring that aspects of the system in consideration are properly described and understood by stakeholders. Multiple cycles of design and evaluation allow to tune the design and fix problems iteratively until all requirements are met. Whereas models and iterative processes are in use, a dangling question remains: how to ensure that models and artifacts remain consistent along an iterative development process. In this context, the present work contributes by providing an approach for specifying and testing user requirements in order to keep the consistency of such requirements with core software artifacts commonly used to build interactive systems.

As we highlighted along this thesis, when assessing software artifacts, the term “test” is usually not employed under the argument that such artifacts cannot be “run”, i.e. executed for testing purposes, so in practice they are just manually reviewed or inspected in a process called verification. Manual verification of the software outcomes (which include modeling artifacts, documentation, source code, database, the product design, etc.) is though highly time-consuming, error-prone and even impracticable for large software systems. Fully interactive artifacts such as final user interfaces can in addition be validated by users who can interact with the artifact and assess whether its behavior is aligned with their actual needs. As within our approach we succeed automatically running User Stories on software artifacts for assessing their consistency with user requirements, we actually provide the “test” component for both verification and validation of artifacts in the software development. We consider this is a big step towards the automated testing (and not only the manual verification) of software artifacts by means of a consistent approach allowing fully verification, validation, and testing (VV&T).

For supporting our approach, an ontology was provided to act as a base of concepts shared by different artifacts, defining a semantic description of user-system interactions. The proposed ontology provides a common vocabulary that is articulated to map interactive behaviors to interaction elements, allowing testing automation of user requirements in multiple user interface design artifacts. The ontology also supports the design of user interfaces by providing a consistent set of interaction elements that are supposed to meet particular behaviors. When representing the behaviors that each interaction element is able to answer, the ontology also allows extending multiple design solutions for the user interface.

Behaviors described in the ontology are already implemented for automating tests on UIs, which means we can freely reuse them to write new scenarios in natural language, providing test automation with little effort from development teams. It allows specifying tests in a generic way, which benefits reuse along the development process. In practice, the vocabulary of interactive behaviors in the ontology also extends the vocabulary provided by the Gherkin DSL (which is implemented by BDD), so indeed we increase the power of expressivity of such a language to automate the assessment of software artifacts. The concepts and definitions in the ontology presented here are nevertheless just one of the possible solutions for addressing and describing behaviors and their relationships with user interface elements. The ontology is provided ready to use for a new development project, but it is not changeless and could be updated with other behaviors, concepts and relationships which could eventually be more representative for some contexts of development. This fact opens the door to consider having ontologies as knowledge bases, keeping specific behaviors for specific groups of business models.

Being core modeling artifacts to design user interfaces, in this thesis we have focused on the assessment of task models, user interface prototypes and final UIs. Compared with co-execution approaches which require a high-level of effort for annotating and modifying the source code/files of the artifacts to make them support automated assessment, we provide a lightweight fully automated approach which provide assessment with no intervention in the source files of the artifacts. This solution allowed us to test prototypes at different stages of the design process, especially from the early phases, following their cycle of evolution and successive refinements, while ensuring that different artifacts are sharing the same goals in terms of requirements. Additionally, tests on web final UIs can run independently of the frameworks used to build the presentation layer. This is possible because tests provided by our tool assess the concrete UI elements found on the user interface, directly in the target web browser, simulating a real user interacting with the interface.

To benefit the testing integration, our approach makes artifacts considered for testing intrinsically related in such a way that a modification in the User Stories scenarios requires a full regression battery of tests in the other artifacts. It is this characteristic that makes our approach ensures fully consistent artifacts for modeling user requirements. As stated before, User Stories in our approach are the main source of requirements and tests, so BDD scenarios are the core of testing. To have a given artifact consistent with them, we should identify the source of inconsistency and fix it either in the artifact or in the BDD step. In principle, if we fix inconsistencies in the artifact which is failing, there is no additional impact in other artifacts. Otherwise, if we keep the design of the artifact and fix the step, we can introduce fails in other artifacts, that are also being tested by the same stories. Thus, it is imperative to work with the perspective of regression tests in mind, which reinforce the crucial importance of automated testing.

Our approach could also be extended to verify and validate other model-based artifacts, allowing more integration and ensuring a wider traceability of requirements. The degree of formality of such artifacts, however, can influence the process of traceability and testing, making it either more or less tricky to conduct. These variations should be investigated in the future.

9.1. Tackled Challenges

Based on the strategy we defined for implementing this approach, we have set out in chapter 3 four main challenges to accomplish it. They are listed as follows:

- (i) *To adhere to a model-based approach for describing artifacts produced along the development process.*
- (ii) *Teams must be willing to adopt the template for User Stories as well as the vocabulary proposed in the ontology.*
- (iii) *Artifacts and the user interface under testing must comply with the UI-supported set of interactive behaviors described in the ontology.*
- (iv) *Tests must be carried out by our set of tools.*

With the results we have obtained and addressing these four challenges we stated above, we can highlight a set of advantages and some shortcomings we have identified so far. Concerning the adherence to a model-based approach, this approach benefits from the independence for testing artifacts. Once the approach performs a micro-process, theoretically suited to run with any macro software development process, testing can be conducted in an independent manner, only in the set of artifacts designed at a given time, which benefits early artifacts. However, so far, we are only covering artifacts modeled with HAMSTERS and Balsamiq. We also did not evaluate yet the impact of maintaining and evolving such artifacts throughout the development process.

Concerning the adoption of the template for User Stories and the vocabulary proposed in the ontology, an advantage is that requirements and tests in User Stories are kept in a natural and high-level language. Keeping them as such helps to establish a common vocabulary for the whole team and allows non-technical stakeholders to effectively participate at the specification and testing processes. Although the studies we have conducted so far did not cover evaluation with potential users instantiating the approach, we plan to investigate its use in a broader case study to evaluate aspects such as workload, maintainability and scalability.

Concerning the expressiveness of the ontology and the compliance of artifacts and user interfaces with the UI-supported set of interactive behaviors, an advantage is that the approach is domain-independent, once the low-level interactive actions on UI elements (such as clicks, selections, settings, etc.) are the same regardless the application business domain. So far, we are applying our ontology to cover the assessment of GUI-based/web-based applications. As the ontology already describes the concepts related to mobile user interfaces, its implementation for covering the assessment of mobile applications is expected to be straightforward. Nonetheless, for covering other types of interaction techniques (such as multimodal interaction), the ontology would need to be extended to model the new concepts related to user interfaces and user-system interactive behaviors on such new environments.

Another advantage of the ontology is the plurality of interaction elements modeled by the ontology used. As many of them can answer the same behavior, even if a Combo Box has been chosen to attend some behavior in a previous prototype, an Auto Complete field could be chosen to attend this behavior on a further and more refined version, once both elements share the same ontological property for the behavior under testing. A shortcoming we have identified is related to the restricted vocabulary of the ontology. Even with the ontology mapping synonyms for some specific behaviors, it does not provide any kind of semantic interpretation, i.e. the behaviors must be specified on stories exactly as they were defined. At a first glance, nonetheless, the restricted vocabulary seems to bring less flexibility to designers, testers and requirements engineers, but at the same time, it establishes a common vocabulary, avoiding typical problems of ambiguity and incompleteness in requirements and testing specifications.

Finally, concerning our tools, one of the advantages they provide is the fine-grained testing coverage. Each small modification in the User Stories or in the artifacts is able to be captured during the testing process. The use of data-independent scenarios is another advantage. Data can

be specified through data domains to be injected on runtime, or directly in the scenario description. The first strategy is very useful in the beginning of the project, when typically, there are few definitions about representative data for testing. A limitation in our set of tools, however, is the absence of classification for errors. There is currently no automatic distinction between the different reasons of test failure (e.g. UI element not found, behavior not supported, etc.). Such analysis should be made manually by the designer. As shown in the case study, our approach signalizes in which step of the scenario some inconsistency has been found, but do not classify it according to the solution that should be employed to solve the problem. Classifying errors would help to better identify if a given inconsistency detected is due to an actual error in the requirements representation or if it is due just to a limitation of the artifact.

9.2. Summary of Contributions

A summary of the thesis contributions is presented as follows:

- A scenario-based approach that benefits from the independence for testing model-based artifacts (chapter 3).
- A full and consistent VV&T approach which actually allows running automated tests on artifacts, expanding the possibilities for software verification and validation (chapters 3, 5, 6 and 8).
- A flexible and adaptable micro-process to instantiate the approach, which could fit different macro software development processes (chapter 3).
- A natural and high-level specification language for requirements and test through a User Story template (chapters 3, 4 and 7).
- A common vocabulary to be used by different stakeholders, avoiding typical problems of ambiguity and incompleteness in requirements and testing specifications (chapters 4 and 7).
- A consistent and domain-independent ontology for specifying interaction (chapter 4).
- An extended vocabulary for the Gherkin DSL increasing the power of expressivity of such a language to automate the assessment of user interfaces (chapters 4, 6 and 8).
- Testing provided with no intervention in the source code of the application or in the source file of the artifacts (chapters 5, 6 and 8).
- Automated tools with a fine-grained testing coverage and implementing data-independent scenarios through the use of data providers (chapters 5, 6 and 8).
- A flexible implementation architecture that can support in the future tests using other notations and tools than HAMSTERS (for task models) and Balsamiq (for UI prototypes) by just implementing new classes for mapping the concepts of the ontology to such notations (chapters 5 and 6).
- A fully compatible approach for testing final UIs which is independent from the technology chosen to implement the presentation layer of web sites (chapters 6 and 8).

9.3. Summary of Limitations

A summary of the thesis limitations is presented as follows:

- A limited vocabulary for the ontology with no semantic interpretation (chapter 4).
- Restricted coverage of artifacts, including so far HAMSTERS task models, Balsamiq prototypes and web final UIs (chapters 5 and 6).

- The need of extracting scenarios from task models to perform testing in such artifacts (chapter 5).
- Tools that do not support yet the classification of errors (chapters 5 and 6).
- Unknown impact of maintaining and successively evolving the mentioned artifacts throughout a real software development process (chapter 8).

9.4. Future Research Perspectives

9.4.1. Short Term Perspective

Although the results presented in this thesis are still preliminary, they are quite promising. The current version of the approach opened the door to many interesting research questions which motivate our future works. First of all, in a short term, we are planning to investigate the acceptability of the approach with development teams, including technical and non-technical people. The idea is to evaluate if people involved in the development process of interactive systems are able to employ our approach to specify user requirements with the proposed template and the concepts present in the ontology. We are planning to conduct such empirical studies with developers, requirement engineers, clients and end-users, in order to determine the potential of improvement in the context of multidisciplinary and complex development teams. These studies should also evaluate the effectiveness of the approach and aspects such as workload, maintainability and scalability when running the approach in real cases of software development.

We are also refining our set of tools to better support the creation, visualization and execution of the tests. An important improvement to address as future works concerns the presentation of task model assessment results. Despite being useful to locate exactly where the correspondent tasks have been found, a presentation based on a detailed list of matching tasks, positions and values tends to be hard to read with the growing of the number of scenarios. Additional features to automatically generate charts, such as the one we used to present the results in chapter 8, might probably help designers to evaluate and better analyzing the results.

Another improvement in the task model assessment that is in the pipeline consists in implementing a better strategy to assess “displaying” tasks, i.e. tasks that involve the system displaying a message to the user, or the user checking that such a message has been displayed. In the current implementation, we search for a task named “Display <message>”, but indeed it is not a common practice to describe system messages literally in a task name, so this kind of tasks are never matched with the equivalent steps in the User Stories. Our first strategy in mind is to look for a generic task named “Display message”, for example, and then check the actual message in the object value associated to this task. While this strategy solves the matching of equivalent tasks, if the designer skips informing the actual message when extracting a scenario from the task model, such a task would still be unidentifiable. Anyway, so far it seems to be the best strategy to address such a problem.

9.4.2. Long Term Perspective

Previous works have proposed the use of Model-Driven Architecture (MDA) to support the development of user interfaces (Vanderdonckt, 2005). In a longer run, we also prospect interesting research opportunities in the field of Model-Driven Development (MDD) for obtaining User Stories and consistent UI prototypes directly from task models. Once it is desirable that scenarios in User Stories and scenarios extracted from task models are compatible, an approach aiming at the automated generation of User Stories scenarios from the scenarios extracted from task models could bring promising results. The first reason is that such scenarios

would be generated already compatible with the task models and would dismiss the need of assessing their XML files, whether the reference XML file or the one extracted with scenarios. The second reason is that, supported by the ontology, this approach could generate scenarios already ready to GUI test automation, allowing automated acceptance testing on user interfaces with low level of implementation effort. It would raise a set of research questions related to the adequacy of user requirements specified only as models in MDD approaches. As such, both modeling and modification on user requirements would be made only in the task model, from where scenarios would be automatically generated to run tests on other artifacts, including acceptance tests on the final UI.

Concerning the automated assessment of user interface prototypes, another potential future work consists in investigating the assessment of prototypes designed by UsiXML prototyping tools. As a well-known standard to describe user interfaces, UsiXML provides the User Interface Description Language (UIDL), a unified notation in which our approach could rely on for providing automated assessment of prototypes designed by different tools. Such an adoption could reduce the need of specializing our implementation for supporting different notations each time a new prototyping tool should be covered. UsiXML could also support the future extension of our ontology to implement automated assessment on other interaction environments by providing description of the concepts related to new interaction techniques.

Further studies on Natural Language Processing (NLP) techniques might help to improve the process of requirements and testing specification adding more flexibility to write scenarios that could be semantically interpreted to meet the behaviors described in the ontology. The ontology could also be enriched to recognize variants for the same interactive behavior. This issue is certainly a worthwhile topic for future research. Evaluate the suitability of our approach to reuse tests for assessing multiple user interface design options is another promising future work. We plan to conduct new case studies to collect data about reusability, workload, and degree of adaptability required to use a same group of business scenarios to test different design solutions. Our hypothesis is that scenarios written based on our common ontology can be easily reused to assess different design solutions for systems sharing the same business model.

Finally, other studies including interactions in different contexts beyond the web, especially in mobile platforms, are also planned. In a longer run, we also want to explore idiosyncrasies of interaction techniques and/or platforms to check hypothesis related to the coverage of concepts in the current ontology. Additional work is also necessary to identify potential problems and inconsistencies when manipulating more complex task models and more complex interactive behaviors. Such studies would contribute to increase the ontology expressiveness. Future works should also consider ontologies as knowledge bases, keeping specific behaviors for specific groups of business models in domain ontologies. Domain-specific ontologies could act as a top layer in a multi-layer ontology architecture to allow the use of multiple domain ontologies associated to the current domain-independent ontology, which would remain describing only the fundamental interactive behaviors for a given environment.

9.5. Full List of Resultant Publications

Journals

Silva, T. R., Hak, J.-L. & Winckler, M. (2017). A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces. International Journal of Semantic Computing, 11 (04), pp. 513-539. DOI: <http://doi.org/10.1142/S1793351X17400219>. (Silva, Hak and Winckler, 2017b)

Silva, T. R., Hak, J.-L., Winckler, M. & Nicolas, O. (2017). A Comparative Study of Milestones for Featuring GUI Prototyping Tools. *Journal of Software Engineering and Applications*, 10 (06), pp. 564-589. DOI: <http://doi.org/10.4236/jsea.2017.106031>. (Silva *et al.*, 2017)

Silva, T. R., Hak, J. L. & Winckler, M. (2016). An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development. *Complex Systems Informatics and Modeling Quarterly*, 1 (7), pp. 81-107. DOI: <http://doi.org/10.7250/csinq.2016-7.05>. (Silva, Hak and Winckler, 2016a)

Conferences

Silva, T. R. & Winckler, M. (2017). A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts. In: Proceedings of the XVI Brazilian Symposium on Human Factors in Computing Systems (IHC 2017), pp. 21-30. ACM. DOI: <http://doi.org/10.1145/3160504.3160506>. (Silva and Winckler, 2017)

Silva, T. R., Hak, J. L. & Winckler, M. (2017). A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems. In: 2017 IEEE 11th International Conference on Semantic Computing (ICSC 2017), pp. 250-257. IEEE. DOI: <http://doi.org/10.1109/ICSC.2017.73>. (Silva, Hak and Winckler, 2017a)

Silva, T. R. (2016). Definition of a Behavior-Driven Model for Requirements Specification and Testing of Interactive Systems. In: 2016 IEEE 24th International Requirements Engineering Conference (RE 2016), pp. 444-449. IEEE. DOI: <http://doi.org/10.1109/RE.2016.12>. (Silva, 2016)

Silva, T. R., Hak, J. L. & Winckler, M. (2016). Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development. In: 6th International Working Conference on Human-Centred Software Engineering, and 8th International Working Conference on Human Error, Safety, and System Development (HCSE 2016 and HESSD 2016), pp. 86-107, vol. 9856. Lecture Notes in Computer Science, Springer International Publishing. DOI: http://doi.org/10.1007/978-3-319-44902-9_7. (Silva, Hak and Winckler, 2016b)

Silva, T. R. & Winckler, M. (2016). Towards Automated Requirements Checking Throughout Development Processes of Interactive Systems. In: 2nd Workshop on Continuous Requirements Engineering, 22nd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ). CEUR-WS.org. (Silva and Winckler, 2016)

Silva, T. R., Hak, J. L. & Winckler, M. (2015). A Review of Milestones in the History of GUI Prototyping Tools. In: INTERACT 2015 Adjunct Proceedings: 15th IFIP TC. 13 International Conference on Human-Computer Interaction, pp. 267-279, vol. 22. University of Bamberg Press. (Silva, Hak and Winckler, 2015)

References

- Adzic, G. (2011) *Specification by Example: How Successful Teams Deliver the Right Software*. Manning Publications.
- Agile Alliance* (2018). Available at: <https://www.agilealliance.org> (Accessed: 1 June 2018).
- Ambler, S. (2002) *Agile Modeling: Effective Practices for eXtreme Programming and the Unified Process*. 1st edn. Wiley.
- Ambler, S. W. (2005) *The Agile Unified Process (AUP)*. Available at: <http://www.ambysoft.com/unifiedprocess/agileUP.html> (Accessed: 1 June 2018).
- Ambler, S. W. and Lines, M. (2012) *Disciplined Agile Delivery (DAD): A Practitioner's Guide to Agile Software Delivery in the Enterprise*. IBM Press.
- Anderson, D. J. (2010) *Kanban: Successful Evolutionary Change for Your Technology Business*. Blue Hole Press.
- Annett, J. (2003) 'Hierarchical Task Analysis', in Hollnagel, E. (ed.) *Handbook of Cognitive Task Design*. Lawrence Erlbaum Associates, pp. 17-35.
- Astels, D. (2003) *Test-Driven Development: A Practical Guide*. 1st edn. Prentice Hall.
- Bano, M. and Zowghi, D. (2013) 'User Involvement in Software Development and System Success: A Systematic Literature Review', in *EASE '13: Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, pp. 125-130. doi: 10.1145/2460999.2461017.
- Barboni, E. et al. (2010) 'Beyond Modelling: An Integrated Environment Supporting Co-Execution of Tasks and Systems Models', in *Proceedings of the 2nd ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '10*, pp. 165-174. doi: 10.1145/1822018.1822043.
- Barnett, J. (2017) *State Chart XML (SCXML): State Machine Notation for Control Abstraction*, W3C. Available at: <http://www.w3.org/TR/scxml/>.
- Beaudouin-Lafon, M. and Mackay, W. E. (2000) 'Prototyping Tools and Techniques', in *Prototype Development and Tools*, pp. 1-41.
- Beck, K. et al. (2001) *Manifesto for Agile Software Development*. Available at: <http://agilemanifesto.org> (Accessed: 1 June 2018).
- Beck, K. (2002) *Test Driven Development: By Example*. 1st edn. Addison-Wesley Professional.
- Beck, K. and Andres, C. (2004) *Extreme Programming Explained: Embrace Change*. 2nd edn. Addison-Wesley.
- Bertolino, A. et al. (2006) 'Product Line Use Cases: Scenario-Based Specification and Testing of Requirements', in *Software Product Lines*. Springer, Berlin, Heidelberg, pp. 425-445. doi: https://doi.org/10.1007/978-3-540-33253-4_11.
- Boehm, B. W. (1979) 'Guidelines for Verifying and Validating Software Requirements and Design Specifications', in *Euro IFIP 79*, pp. 711-719. doi: 10.1109/MS.1984.233702.

- Booch, G., Rumbaugh, J. and Jacobson, I. (2005) *The Unified Modeling Language User Guide*. 2nd edn. Addison-Wesley Professional. Available at: <http://portal.acm.org/citation.cfm?id=1088874>.
- Bowen, J. and Reeves, S. (2011) 'UI-Driven Test-First Development of Interactive Systems', in *Proceedings of the 3rd ACM SIGCHI symposium on Engineering interactive computing systems - EICS '11*, pp. 165–174. doi: 10.1145/1996461.1996515.
- Buchmann, R. A. and Karagiannis, D. (2017) 'Modelling mobile app requirements for semantic traceability', *Requirements Engineering*. Springer London, 22(1), pp. 41–75. doi: 10.1007/s00766-015-0235-1.
- Business Process Model And NotationTM (BPMNTM)* (2011) *Object Management Group*. Available at: <http://www.omg.org/spec/BPMN/2.0/> (Accessed: 1 December 2017).
- Calvary, G. et al. (2002) *The CAMELEON Reference Framework, R&D Project IST-2000-30104*. Available at: <http://glove.isti.cnr.it/projects/cameleon.html>.
- Calvary, G. et al. (2003) 'A Unifying Reference Framework for multi-target user interfaces', *Interacting with Computers*, 15(3 SPEC.), pp. 289–308. doi: 10.1016/S0953-5438(03)00010-9.
- Calvary, G., Coutaz, J. and Thevenin, D. (2001) 'Supporting Context Changes for Plastic User Interfaces: A Process and a Mechanism', in *People and Computers XV – Interaction without Frontiers*. Springer, pp. 349–363.
- Campos, J. C. et al. (2016) 'Systematic Automation of Scenario-Based Testing of User Interfaces', in *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '16*, pp. 138–148. doi: 10.1145/2933242.2948735.
- Campos, J. C. et al. (2017) 'A More Intelligent Test Case Generation Approach through Task Models Manipulation', *Proceedings of the ACM on Human-Computer Interaction*, 1(1), pp. 1–20. doi: 10.1145/3095811.
- Card, S. K., Newell, A. and Moran, T. P. (1983) *The Psychology of Human-Computer Interaction*. L. Erlbaum Associates Inc.
- Carvalho, R. A. de, Carvalho e Silva, F. L. de and Manhaes, R. S. (2010) *Mapping Business Process Modeling constructs to Behavior Driven Development Ubiquitous Language*, arXiv:1006.4892. Available at: <http://arxiv.org/abs/1006.4892>.
- Carvalho, R. A. de, Manhães, R. S. and Carvalho e Silva, F. L. de (2010) *Filling the Gap between Business Process Modeling and Behavior Driven Development*, arXiv preprint arXiv:1005.4975. Available at: <http://arxiv.org/abs/1005.4975>.
- Chelimsky, D. et al. (2010) *The RSpec Book: Behaviour Driven Development with RSpec, Cucumber, and Friends*. Pragmatic Bookshelf.
- Chikofsky, E. J. and Cross II, J. H. (1990) 'Reverse Engineering and Design Recovery: A Taxonomy', *IEEE Software*, pp. 13–17. doi: 10.1109/52.43044.
- Cohn, M. (2004) *User Stories Applied for Agile Software Development*. Addison-Wesley.
- Coyette, A., Kieffer, S. and Vanderdonckt, J. (2007) 'Multi-fidelity Prototyping of User Interfaces', in *Proc. of the IFIP TC.13 International Conference on Human-Computer Interaction*, pp. 150–164. doi: 10.1007/978-3-540-74796-3_16.

References

- Crandall, B., Klein, G. and Hoffman, R. R. (2006) *Working Minds: A Practitioner's Guide to Cognitive Task Analysis*. MIT Press.
- Dalkir, K. (2011) *Knowledge Management in Theory and Practice*. MIT Press. doi: 10.1002/asi.21613.
- Dijkstra, E. W. (1970) *On The Reliability of Mechanisms*.
- Dumontier, M. (2018) *Ontology Design Principles*, GitHub. Available at: <https://github.com/micheldumontier/semanticscience/wiki/Ontology-Design-Principles> (Accessed: 6 August 2018).
- Dwarakanath, A. and Sengupta, S. (2012) 'Litmus: Generation of Test Cases from Functional Requirements in Natural Language', in *Int. Conference on Application of Natural Language to Information Systems*, pp. 58–69. doi: 10.1007/978-3-642-31178-9_6.
- Ebert, C. (2011) *Global Software and IT: A Guide to Distributed Development, Projects, and Outsourcing*. John Wiley & Sons.
- Egbreghs, A. (2017) 'A Literature Review of Behavior Driven Development using Grounded Theory', in *27th Twente Student Conference on IT*. Available at: <https://pdfs.semanticscholar.org/4f03/ec0675d08cf1ecdba3361a29d756ce656.pdf>.
- Elkoutbi, M., Khriss, I. and Keller, R. K. (2006) 'Automated Prototyping of User Interfaces Based on UML Scenarios', in *Automated Software Engineering*. Volume 13,. Kluwer Academic Publishers, pp. 5–40. doi: <https://doi.org/10.1007/s10515-006-5465-5>.
- Engel, A. (2010) *Verification, Validation, and Testing of Engineered Systems*. John Wiley & Sons, Inc.
- Fahssi, R., Martinie, C. and Palanque, P. (2015) 'Enhanced Task Modelling for Systematic Identification and Explicit Representation of Human Errors', in *Proc. of the IFIP TC.13 International Conference on Human-Computer Interaction*, pp. 192–212. doi: 10.1007/978-3-319-22723-8.
- Farooq Ali, M., Pérez-Quiñones, M. A. and Abrams, M. (2005) 'Building Multi-Platform User Interfaces with UIML', in Seffah, A. and Javahery, H. (eds) *Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces*. John Wiley & Sons, pp. 93–118. doi: 10.1002/0470091703.ch6.
- Fierstone, J., Dery-Pinna, A.-M. and Riveill, M. (2003) *Architecture Logicielle pour l'adaptation et la composition d'IHM - Mise en œuvre avec le langage SUNML*.
- Forsberg, K. and Mooz, H. (1991) 'The Relationship of System Engineering to the Project Cycle', in *Proceedings of the First Annual Symposium of National Council on System Engineering*, pp. 57–65. doi: 10.1002/j.2334-5837.1991.tb01484.x.
- Graham, D. *et al.* (2008) *Foundations of Software Testing: ISTQB Certification*. Cengage Learning Emea.
- Green, M. (1985) 'Report on Dialogue Specification Tools', in *User Interface Management Systems*. Springer, Berlin, Heidelberg, pp. 9–20. doi: https://doi.org/10.1007/978-3-642-70041-5_2.
- Guarino, N., Oberle, D. and Staab, S. (2009) 'What Is an Ontology?', in *Handbook on Ontologies*. Springer, pp. 1–17.
- Hackos, J. T. and Redish, J. C. (1998) *User and Task Analysis for Interface Design*. 1st edn. John Wiley & Sons, Inc.
- Hak, J., Winckler, M. and Navarre, D. (2016) 'PANDA : Prototyping using ANnotation and Decision

- Analysis', in *Proceedings of the 8th ACM SIGCHI Symposium on Engineering Interactive Computing Systems*, pp. 171–176.
- Hellmann, T. D. (2015) *Automated GUI Testing for Agile Development Environments*. University of Calgary.
- Highsmith, J. R. (1999) *Adaptive Software Development: A Collaborative Approach to Managing Complex Systems*. Dorset House.
- Hotomski, S., Charrada, E. Ben and Glinz, M. (2017) 'Aligning Requirements and Acceptance Tests via Automatically Generated Guidance', in *2017 IEEE 25th International Requirements Engineering Conference Workshops*, pp. 339–342. doi: 10.1109/REW.2017.87.
- IEEE (2017) *IEEE Standard for System and Software Verification and Validation IEEE*. IEEE Computer Society. doi: 10.1109/IEEESTD.2012.6204026.
- ISO (1999) *ISO 13407: Human-centred design processes for interactive systems*.
- Jacobson, I., Booch, G. and Rumbaugh, J. (1999) *The Unified Software Development Process*. 1st edn. Addison-Wesley Professional.
- Käpyaho, M. and Kauppinen, M. (2015) 'Agile Requirements Engineering With Prototyping: A Case Study', in *2015 IEEE 23rd International Requirements Engineering Conference, RE 2015 - Proceedings*, pp. 334–343. doi: 10.1109/RE.2015.7320450.
- Khaddam, I., Mezhoudi, N. and Vanderdonckt, J. (2015) 'Towards Task-Based Linguistic Modeling for Designing GUIs', in *27th Conference on l'Interaction Homme-Machine*.
- Ladas, C. (2009) *Scrumban - Essays on Kanban Systems for Lean Software Development*. Modus Cooperandi Press.
- Lai, S.-T., Leu, F.-Y. and Chu, W. C.-C. (2014) 'Combining IID with BDD to Enhance the Critical Quality of Security Functional Requirements', in *2014 Ninth International Conference on Broadband and Wireless Computing, Communication and Applications (BWCCA)*. IEEE.
- Landay, J. A. (1996) 'SILK: Sketching Interfaces Like Krazy', in *CHI 96*, pp. 398–399. doi: 10.1145/257089.257396.
- Leite, J. C. S. do P. and Oliveira, A. D. P. A. (1995) 'A Client Oriented Requirements Baseline', in *International Conference on Requirements Engineering*, pp. 108–115. doi: 10.1109/ISRE.1995.512551.
- Lewis, C. and Rieman, J. (1993) *Task-Centered User Interface Design: A Practical Introduction*. doi: 10.1017/CBO9781107415324.004.
- Limbourg, Q. et al. (2004) 'USIXML: A Language Supporting Multi-path Development of User Interfaces', in *EHCI/DS-VIS*, pp. 200–220. doi: 10.1007/11431879_12.
- Limbourg, Q. and Vanderdonckt, J. (2003) 'Comparing Task Models for User Interface Design', in Diaper, D. and Stanton, N. (eds) *The Handbook of Task Analysis for Human-Computer Interaction*. Taylor & Francis, pp. 135–154. doi: 10.1.1.58.1444.
- Lindstrom, D. R. (1993) 'Five Ways to Destroy a Development Project', *IEEE Software*, 10(5), pp. 55–58. doi: 10.1109/52.232400.
- Lombriser, P. et al. (2016) 'Gamified Requirements Engineering: Model and Experimentation', in

Proceedings of the 22nd International Working Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2016). Springer-Verlag Berlin, pp. 171–187.

Lopes, J. H. (2012) *Evaluation of Behavior-Driven Development*. Delft University of Technology.

Lübke, D. and Van Lessen, T. (2016) ‘Modeling Test Cases in BPMN for Behavior- Driven Development’, *IEEE Software*, (October), pp. 15–21. doi: 10.1109/MS.2016.117.

Lucassen, G. et al. (2017) ‘Behavior-Driven Requirements Traceability via Automated Acceptance Tests’, *Proceedings - 2017 IEEE 25th International Requirements Engineering Conference Workshops, REW 2017*, pp. 431–434. doi: 10.1109/REW.2017.84.

Luna, E. R. et al. (2010) ‘Capture and Evolution of Web Requirements Using WebSpec’, in *Proc. of the Int. Conference on Web Engineering*, pp. 173–188. doi: 10.1007/978-3-642-13911-6_12.

Maguire, M. and Bevan, N. (2002) ‘User Requirements Analysis: A Review of Supporting Methods’, in *IFIP World Computer Congress*. Kluwer Academic Publishers, pp. 133–148.

Marcotte, E. (2014) *Responsive Web Design*. 2nd Editio. A Book Apart, LLC.

Martin, J. (1991) *Rapid Application Development*. Macmillan Publishing Co.

Martinie, C. et al. (2013) ‘Extending Procedural Task Models by Systematic Explicit Integration of Objects, Knowledge and Information’, in *Proceedings of the 31st European Conference on Cognitive Ergonomics*, p. European Association for Cognitive Ergonomics (EAC). doi: 10.1145/2501907.2501954.

Martinie, C. et al. (2015) ‘A Generic Tool-Supported Framework for Coupling Task Models and Interactive Applications’, in *Proceedings of the 7th ACM SIGCHI Symposium on Engineering Interactive Computing Systems - EICS '15*, pp. 244–253. doi: 10.1145/2774225.2774845.

Martinie, C., Palanque, P. and Winckler, M. (2011) ‘Structuring and Composition Mechanisms to Address Scalability Issues in Task Models’, in *Proc. of the IFIP TC.13 International Conference on Human-Computer Interaction*, pp. 589–609. doi: 10.1007/978-3-642-23765-2_40.

McDonald, J. E., Vandenberg, P. D. J. and Smartt, M. J. (1988) ‘The mirage rapid interface prototyping system’, in *UIST '88 Proceedings of the 1st annual ACM SIGGRAPH symposium on User Interface Software*, pp. 77–84.

van Megen, R. and Meyerhoff, D. B. (1995) ‘Costs and benefits of early defect detection: experiences from developing client server and host applications’, *Software Quality Journal*, 4(4), pp. 247–256. doi: 10.1007/BF00402646.

Melnik, G. I. (2007) *Empirical Analyses of Executable Acceptance Test Driven Development*. University of Calgary.

Meyer, B. (1985) ‘On Formalism in Specifications’, *IEEE Software*, 2(1), pp. 6–26. doi: 10.1109/MS.1985.229776.

Myers, G. J. (2004) *The Art of Software Testing*. 2nd edn. John Wiley & Sons, Inc.

Nair, S., De La Vara, J. L. and Sen, S. (2013) ‘A Review of Traceability Research at the Requirements Engineering Conference RE@21’, in *2013 21st IEEE International Requirements Engineering Conference, RE 2013 - Proceedings*, pp. 222–229. doi: 10.1109/RE.2013.6636722.

Navarre, D. et al. (2001) ‘A Tool Suite for Integrating Task and System Models Through Scenarios’, in

DSV-IS, pp. 88–113. doi: 10.1007/3-540-45522-1_6.

Navarre, D., Palanque, P. and Bastide, R. (2002) ‘Model-Based Interactive Prototyping of Highly Interactive Applications’, in *Computer-Aided Design of User Interfaces III*. Springer, pp. 205–216.

Newman, M. et al. (2003) ‘DENIM: An Informal Web Site Design Tool Inspired by Observations of Practice’, *Human-Computer Interaction*, 18(3), pp. 259–324. doi: 10.1207/S15327051HCI1803_3.

Nicolle, C. A. (1999) ‘USERfit - Design for all methods and tools’, in *COST 219bis Seminar ‘Human Aspects of Telecommunications for Disabled and Older People’*. Donostia-San Sebastian, Spain, Spain. Available at: <http://hdl.handle.net/2134/1026>.

Nielsen, J. (1986) ‘A Virtual Protocol Model for Computer-Human Interaction’, *International Journal of Man-Machine Studies*, 24(3), pp. 301–312. doi: 10.1016/S0020-7373(86)80028-1.

North, D. (2006) ‘Introducing BDD’, *Better Software*.

North, D. (2009) ‘Agile Specifications, BDD and Testing eXchange: How to sell BDD to the business’. London: Skills Matter. Available at: <https://skillsmatter.com/skillscasts/923-how-to-sell-bdd-to-the-business#video>.

North, D. (2017) *What’s in a Story?* Available at: <https://dannorth.net/whats-in-a-story/> (Accessed: 1 December 2017).

Oran, A. C. et al. (2017) ‘Analysing Requirements Communication Using Use Case Specification and User Stories’, in *Proceedings of the 31st Brazilian Symposium on Software Engineering (SBES 2017)*. ACM, pp. 214–223. doi: <https://doi.org/10.1145/3131151.3131166>.

Osterweil, L. J. (2005) ‘Unifying Microprocess and Macroprocess Research’, in *Unifying the Software Process Spectrum*. Springer, Berlin, Heidelberg, pp. 68–74. doi: https://doi.org/10.1007/11608035_7.

Palmer, S. R. and Felsing, J. M. (2002) *A Practical Guide to Feature-Driven Development*. 1st edn. Prentice Hall.

Paternò, F. (1999) ‘ConcurTaskTrees : An Engineered Approach to Model-based Design of Interactive Systems’, in *The Handbook of Analysis for Human Computer Interaction*, pp. 1–18. doi: 10.1111/j.1467-923X.1954.tb00152.x.

Paternò, F. (2000) *Model-Based Design and Evaluation of Interactive Applications*. Springer-Verlag London.

Paternò, F. (2003) ‘ConcurTaskTrees: An Engineered Notation for Task Model’, in *The Handbook of Task Analysis for Human-Computer Interaction*. Lawrence Erlbaum Associates, pp. 483–503.

Paternò, F. et al. (2017) *W3C, MBUI - Task Models*. Available at: <http://www.w3.org/TR/task-models/>.

Paternò, F. and Mancini, C. (1999) ‘Developing task models from informal scenarios’, in *CHI EA ’99 CHI ’99 Extended Abstracts on Human Factors in Computing Systems*. ACM, pp. 228–229. doi: <https://doi.org/10.1145/632716.632858>.

Pontico, F., Farenc, C. and Winckler, M. (2007) ‘Model-Based Support for Specifying eService eGovernment Applications’, in *Task Models and Diagrams for Users Interface Design: 5th International Workshop, TAMODIA 2006, Hasselt, Belgium, October 23-24, 2006. Revised Papers*, pp. 54–67. doi: 10.1007/978-3-540-70816-2_5.

- Poppendieck, M., Poppendieck, T. D. and Poppendieck, T. (2003) *Lean Software Development: An Agile Toolkit*. Addison-Wesley Professional.
- Puerta, A. and Eisenstein, J. (2002) 'XIML: A Universal Language for User Interfaces', in *Proceedings of the 7th International Conference on Intelligent User Interfaces*. Available at: <http://www.ximl.org/documents/XimlWhitePaper.pdf>.
- Pugh, K. (2010) *Lean-Agile Acceptance Test-Driven-Development*. Pearson Education.
- Pullmann, J. (2017) *MBUI - Glossary - W3C*. Available at: <https://www.w3.org/TR/mbui-glossary/> (Accessed: 1 December 2017).
- Rahman, M. and Gao, J. (2015) 'A Reusable Automated Acceptance Testing Architecture for Microservices in Behavior-Driven Development', in *Proceedings - 9th IEEE International Symposium on Service-Oriented System Engineering, IEEE SOSE 2015*, pp. 321-325. doi: 10.1109/SOSE.2015.55.
- Ramesh, B. et al. (1995) 'Implementing Requirements Traceability: A Case Study', in *Proceedings of the Second IEEE International Symposium on Requirements Engineering*. York, United Kingdom, pp. 89-95.
- Reddy, A. (2015) *Scrumban /R/Evolution, The: Getting the Most Out of Agile, Scrum, and Lean Kanban*. 1st edn. Addison-Wesley Professional.
- Rosson, M. B. and Carroll, J. M. (2001) *Usability Engineering: Scenario-Based Development of Human-Computer Interaction*. Morgan Kaufmann.
- Rosson, M. B. and Carroll, J. M. (2002) 'Scenario-Based Design', in *The Human-Computer Interaction Handbook: Fundamentals, Evolving Technologies and Emerging Applications*, pp. 1032-1050. doi: 10.1016/j.jbhi.2011.07.004.
- Royce, D. W. W. (1970) 'Managing the Development of Large Software Systems', *IEEE Wescon*, (August), pp. 328-338.
- Saffer, D. (2006) *Designing for Interaction: Creating Smart Applications and Clever Devices*. 1st edn. New Riders.
- Santoro, C. (2005) *A Task Model-Based Approach for the Design and Evaluation of Innovative User Interfaces*. Consiglio Nazionale Delle Ricerche.
- Schwaber, K. (2004) *Agile Project Management with Scrum*. Microsoft Press.
- Silva, T. R. (2016) 'Definition of a Behavior-Driven Model for Requirements Specification and Testing of Interactive Systems', in *Proceedings - 2016 IEEE 24th International Requirements Engineering Conference, RE 2016*, pp. 444-449. doi: 10.1109/RE.2016.12.
- Silva, T. R. et al. (2017) 'A Comparative Study of Milestones for Featuring GUI Prototyping Tools', *Journal of Software Engineering and Applications*, 10(06), pp. 564-589. doi: 10.4236/jsea.2017.106031.
- Silva, T. R., Hak, J.-L. and Winckler, M. (2016a) 'An Approach for Multi-Artifact Testing Through an Ontological Perspective for Behavior-Driven Development', *Complex Systems Informatics and Modeling Quarterly*, (7), pp. 81-107. doi: 10.7250/csimq.2016-7.05.
- Silva, T. R., Hak, J.-L. and Winckler, M. (2016b) 'Testing Prototypes and Final User Interfaces Through an Ontological Perspective for Behavior-Driven Development', in *6th International Working Conference on Human-Centred Software Engineering, and 8th International Working Conference on Human Error*,

Safety, and System Development (HCSE 2016 and HESSD 2016), pp. 86–107. doi: 10.1007/978-3-319-44902-9.

Silva, T. R., Hak, J.-L. and Winckler, M. (2017a) ‘A Behavior-Based Ontology for Supporting Automated Assessment of Interactive Systems’, in *Proceedings - IEEE 11th International Conference on Semantic Computing, ICSC 2017*, pp. 250–257. doi: 10.1109/ICSC.2017.73.

Silva, T. R., Hak, J.-L. and Winckler, M. (2017b) ‘A Formal Ontology for Describing Interactive Behaviors and Supporting Automated Testing on User Interfaces’, *International Journal of Semantic Computing*, 11(04), pp. 513–539. doi: 10.1142/S1793351X17400219.

Silva, T. R., Hak, J.-L. and Winckler, M. A. (2015) ‘A Review of Milestones in the History of GUI Prototyping Tools’, in *IFIP TC.13 International Conference on Human-Computer Interaction - INTERACT 2015 Adjunct Proceedings*, pp. 1–13.

Silva, T. R. and Winckler, M. (2017) ‘A Scenario-Based Approach for Checking Consistency in User Interface Design Artifacts’, in *IHC'17, Proceedings of the 16th Brazilian Symposium on Human Factors in Computing Systems*, pp. 21–30. doi: 10.1145/3160504.3160506.

Silva, T. R. and Winckler, M. A. A. (2016) ‘Towards Automated Requirements Checking Throughout Development Processes of Interactive Systems’, in *22nd International Working Conference on Requirements Engineering - Foundation for Software Quality, REFSQ 2016*, pp. 1–2.

Sneed, H. M. (2007) ‘Testing against Natural Language Requirements’, in *Proc. of the Seventh International Conference on Quality Software*, pp. 380–387. doi: 10.1109/QSIC.2007.4385524.

Soeken, M., Wille, R. and Drechsler, R. (2012) ‘Assisted Behavior Driven Development Using Natural Language Processing’, in *TOOLS Europe 2012*, pp. 269–287. Available at: <http://www.scopus.com/inward/record.url?eid=2-s2.0-84862188111&partnerID=40&md5=3f9e6990f0071f032a96c05dea733985>.

Solis, C. and Wang, X. (2011) ‘A Study of the Characteristics of Behaviour Driven Development’, in *Proceedings - 37th EUROMICRO Conference on Software Engineering and Advanced Applications, SEAA 2011*, pp. 383–387. doi: 10.1109/SEAA.2011.76.

Sousa, K. S. K., Mendonça, H. and Vanderdonckt, J. (2008) ‘A Model-Driven Approach to Align Business Processes with User Interfaces’, *Journal of Universal Computer Science*, 14(19), pp. 3236–3249.

De Souza, C. S. (2005) ‘Semiotic engineering: Bringing designers and users together at interaction time’, *Interacting with Computers*, 17(3), pp. 317–341. doi: 10.1016/j.intcom.2005.01.007.

Stapleton, J. and Constable, P. (1997) *DSDM: Dynamic Systems Development Method: The Method in Practice*. 1st edn. Addison-Wesley Professional.

Tian, J. (2005) ‘Software Inspection’, in *Software Quality Engineering: Testing, Quality Assurance, and Quantifiable Improvement*. John Wiley & Sons, Inc., pp. 237–250. doi: <https://doi.org/10.1002/0471722324.ch14>.

Usability Body of Knowledge (2018) *User Experience Professionals’ Association*. Available at: <http://www.usabilitybok.org> (Accessed: 1 June 2018).

Uusitalo, E. J. et al. (2008) ‘Linking Requirements and Testing in Practice’, in *Proceedings of the 16th IEEE International Requirements Engineering Conference, RE'08*, pp. 265–270. doi: 10.1109/RE.2008.30.

- Valente, P. *et al.* (2016) 'Bridging Enterprise and Software Engineering Through an User-Centered Design Perspective', in *Web Information Systems Engineering - WISE 2016*, pp. 349–357. doi: 10.1007/978-3-319-48743-4.
- Valente, P. *et al.* (2017) 'The Goals Approach: Agile Enterprise Driven Software Development', in *Complexity in Information Systems Development*, pp. 201–219. doi: 10.1007/978-3-319-52593-8.
- Vanderdonckt, J. (2005) 'A MDA-Compliant Environment for Developing User Interfaces of Information Systems', in *CAiSE 2005*, pp. 16–31. doi: 10.1007/11431855_2.
- Wang, Y. and Wagner, S. (2018) 'Combining STPA and BDD for Safety Analysis and Verification in Agile Development: A Controlled Experiment', in *International Conference on Agile Software Development (XP 2018)*. Springer, pp. 37–53. doi: https://doi.org/10.1007/978-3-319-91602-6_3.
- Wautelet, Y. *et al.* (2014) 'Unifying and Extending User Story Models', in *International Conference on Advanced Information Systems Engineering (CAiSE 2014)*. Springer, pp. 211–225. doi: 10.1007/978-3-319-07881-6_15.
- Winckler, M. *et al.* (2008) 'Cascading dialog modeling with UsiXML', in *International Workshop on Design, Specification, and Verification of Interactive Systems*, pp. 121–135.
- Winckler, M. and Palanque, P. (2003) 'StateWebCharts: A Formal Description Technique Dedicated to Navigation Modelling of Web Applications', in *Interactive Systems. Design, Specification, and Verification*, pp. 61–76. doi: 10.1007/978-3-540-39929-2_5.
- Winckler, M. and Palanque, P. (2012) 'Models as Representations for Supporting the Development of e-Procedures', in *Usability in Government Systems*. Elsevier, pp. 301–315. doi: 10.1016/B978-0-12-391063-9.00051-1.
- Wolff, A. *et al.* (2005) 'Linking GUI Elements to Tasks – Supporting an Evolutionary Design Process', in *Proceedings of the 4th International Workshop on Task Models and Diagrams*, pp. 27–34. doi: 10.1145/1122935.1122941.
- Wood, D. P. and Kang, K. C. (1992) *A Classification and Bibliography of Software Prototyping, Requirements Engineering Project*. Pittsburgh, Pennsylvania. Available at: <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.6660&rep=rep1&type=pdf>.

Appendix A: Concept Mapping Table

Checkbox and Radio Button Behaviors					
Ontological Behavior	Task	Step of Scenario	Interactive Elements Affected		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>theFieldIsUnchecked</i>	<i>Verify the field <fieldname> is unchecked</i>	<i>Given/When/Then the field "<fieldname>" is unchecked</i>	Checkbox	CheckBox	CheckBox
			Radio Button	RadioButton	Radio
<i>theFieldIsChecked</i>	<i>Verify the field <fieldname> is checked</i>	<i>Given/When/Then the field "<fieldname>" is checked</i>	Checkbox	CheckBox	CheckBox
			Radio Button	RadioButton	Radio
<i>assureTheFieldIsUnchecke d</i>	<i>Assure the field <fieldname> is unchecked</i>	<i>When I assure the field "<fieldname>" is unchecked</i>	Checkbox	CheckBox	CheckBox
<i>assureTheFieldIsChecked</i>	<i>Assure the field <fieldname> is checked</i>	<i>When I assure the field "<fieldname>" is checked</i>	Checkbox	CheckBox	CheckBox
Common Behaviors					
Ontological Behavior	Task	Step of Scenario	Interactive Elements Affected		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>choose</i>	<i>Choose <option></i>	<i>Given/When/Then I choose "<option>"</i>	Calendar	Calendar or DateChooser	Calendar
			Checkbox	CheckBox	CheckBox
			Radio Button	RadioButton	Radio
			Link	Link	Link
<i>select</i>	<i>Select <option></i>	<i>Given/When/Then I select "<option>"</i>	Calendar	Calendar or DateChooser	Calendar
			Checkbox	CheckBox	CheckBox
			Radio Button	RadioButton	Radio
			Link	Link	Link

Appendix A: Concept Mapping Table

<i>chooseByIndexInTheField</i>	<i>Choose in the field <fieldname></i>	<i>When/Then I choose "<index>" by index in the field "<fieldname>"</i>	Dropdown List	ComboBox	Select
<i>chooseReferringTo</i>	<i>Choose <fieldname> referring to <option></i>	<i>When/Then I choose "<fieldname>" referring to "<option>"</i>	Calendar	Calendar or DateChooser	Calendar
	<i>Choose <fieldname></i>		Checkbox	CheckBox	CheckBox
			Radio Button	RadioButton	Radio
			Link	Link	Link
<i>chooseTheOptionOfValueInTheField</i>	<i>Choose in the field <fieldname></i>	<i>When/Then I choose the option of value "<value>" in the field "<fieldname>"</i>	Dropdown List	ComboBox	Select
<i>clickOn</i>	<i>Click on <fieldname></i>	<i>When/Then I click on "<fieldname>"</i>	Menu	MenuBar	Menu
			MenuItem	Accordion	MenuItem
			Button	Button	Button
			Link	Link	Link
<i>clickOnReferringTo</i>	<i>Click on <fieldname> referring to <option></i>	<i>When/Then I click on "<fieldname>" referring to "<option>"</i>	Menu	MenuBar	Menu
	MenuItem		Accordion	MenuItem	
	Button		Button	Button	
	Link		Link	Link	
	<i>Click on <fieldname></i>		Grid	DataGrid	Grid
<i>doNotTypeAnyValueToTheField</i>	<i>Do not type any value to the field <fieldname></i>	<i>When I do not type any value to the field "<fieldname>"</i>	Text Field	TextInput	TextField
<i>resetTheValueOfTheField</i>	<i>Reset the value of the field <fieldname></i>	<i>When I reset the value of the field "<fieldname>"</i>	Text Field	TextInput	TextField
<i>goTo</i>	<i>Go to <address></i>	<i>Given/When/Then I go to "<address>"</i>	Browser Window	BrowserWindow	Screen
<i>goToWithTheParameters</i>	<i>Go to <address> with the parameters <parameters></i>	<i>Given/When/Then I go to "<address>" with the parameters "<parameters>"</i>	Browser Window	BrowserWindow	Screen
<i>isDisplayed</i>	<i>Display <page></i>	<i>Given/When/Then "<page>" is displayed</i>	Browser Window	BrowserWindow	Screen
<i>setInTheField</i>	<i>Set <fieldname></i>	<i>When/Then I set "<value>" in the field "<fieldname>"</i>	Dropdown List	ComboBox	Select
			Text Field	TextInput	TextField
			Autocomplete	SearchBox	AutoComplete

Appendix A: Concept Mapping Table

			Calendar	Calendar or DateChooser	Calendar
tryToSetInTheField	Try to set <fieldname>	When/Then I try to set in the field “<fieldname>”	Dropdown List	ComboBox	Select
			Text Field	TextInput	TextField
			Autocomplete	SearchBox	AutoComplete
			Calendar	Calendar or DateChooser	Calendar
setInTheFieldReferringTo	Set <fieldname>	When/Then I set “<value>” in the field referring to “fieldname”	Dropdown List	ComboBox	Select
			Text Field	TextInput	TextField
typeAndChooseInTheField	Inform <value 1> Choose <value 2>	When/Then I type “<value 1>” and choose “<value 2>” in the field “<fieldname>”	Autocomplete	SearchBox	AutoComplete
informAndChooseInTheField	Inform <value 1> Choose <value 2>	When/Then I inform “<value 1>” and choose “<value 2>” in the field “<fieldname>”	Autocomplete	SearchBox	AutoComplete
willBeDisplayed	Display <content>	Then “<content>” will be displayed	Text	Paragraph	Text
willNotBeDisplayed	Not display <content>	Then “<content>” will not be displayed	Text	Paragraph	Text
willBeDisplayedInTheFieldTheValue	Display <value>	Then will be displayed in the field “<fieldname>” the value “<value>”	Element	UI Element	Element
willNotBeDisplayedInTheFieldTheValue	Not display <value>	Then will not be displayed in the field “<fieldname>” the value “<value>”	Element	UI Element	Element
willBeDisplayedTheValueInTheFieldReferringTo	Display <value>	Then will be displayed the value “<value>” in the field “<fieldname>” referring to “<element>”	Element	UI Element	Element
willNotBeDisplayedTheValueInTheFieldReferringTo	Not display <value>	Then will not be displayed the value “<value>” in the field “<fieldname>” referring to “<element>”	Element	UI Element	Element
isNotVisible	Hidden <fieldname>	Given/When/Then “<fieldname>” is not visible	Element	UI Element	Element
valueReferringToIsNotVisible	Hidden <value>	Given/When/Then “<value>” referring to “<element>” is not visible	Element	UI Element	Element

Appendix A: Concept Mapping Table

<i>waitTheFieldBeVisibleClickableAndEnable</i>	<i>Wait the field <fieldname> be visible, clickable and enable</i>	<i>Given/When/Then I wait the field “<fieldname>” be visible, clickable and enable</i>	Element	UI Element	Element
<i>waitTheFieldReferringToBeVisibleClickableAndEnable</i>	<i>Wait the field <fieldname> be visible, clickable and enable</i>	<i>Given/When/Then I wait the field “<fieldname>” referring to “<element>” be visible, clickable and enable</i>	Element	UI Element	Element
<i>theElementIsVisibleAndDisable</i>	<i>Check the element <element> is visible and disable</i>	<i>Given/When/Then the element “<element>” is visible and disable</i>	Element	UI Element	Element
<i>theElementReferringToIsVisibleAndDisable</i>	<i>Check the element <element> is visible and disable</i>	<i>Given/When/Then the element “<fieldname>” referring to “<element>” is visible and disable</i>	Element	UI Element	Element
<i>setInTheFieldAndTriggerTheEvent</i>	<i>Set <fieldname> Trigger <event></i>	<i>When/Then I set in the field “<fieldname>” and trigger the event “<event>”</i>	Text Field	TextInput	TextField
<i>clickOnTheRowOfTheTree</i>	<i>Select value for <tree></i>	<i>Given/When/Then I click on the row “<row>” of the tree “<tree>”</i>	Tree	-	Tree
Data Generation Behaviors					
Ontological Behavior	Task	Step of Scenario	Interactive Elements Affected		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>informARandomNumberWithPrefixInTheField</i>	<i>Inform a random number with prefix in the field <fieldname></i>	<i>Given/When/Then I inform a random number with prefix “<prefix>” in the field “<fieldname>”</i>	Text Field	TextInput	TextField
<i>informARandomNumberInTheField</i>	<i>Inform a random number in the field <fieldname></i>	<i>When I inform a random number in the field “<fieldname>”</i>	Text Field	TextInput	TextField
Data Provider Behaviors					
Ontological Behavior	Task	Step of Scenario	UI Elements		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>inform</i>	<i>Inform <value></i>	<i>Given/When I inform “<value>”</i>	Grid	DataGrid	Grid

Appendix A: Concept Mapping Table

<i>informTheField</i>	<i>Inform the field <fieldname></i>	When I inform the field “<fieldname>”	Grid	DataGrid	Grid
<i>informTheFields</i>	<i>Inform the fields <fieldnames></i>	When I inform the fields “<fieldnames>”	Grid	DataGrid	Grid
<i>selectFromDataSet</i>	<i>Select from dataset <dataset></i>	Given/When I select from dataset “<dataset>”	-	-	-
<i>informTheValueOfTheField</i>	<i>Inform the value of the field <fieldname></i>	When/Then I inform the value of the field “<fieldname>”	Element	UI Element	Element
<i>informKeyWithTheValue</i>	<i>Inform key <key></i>	Given/When/Then I inform key “<key>” with the value “<value>”	-	-	-
<i>defineTheVariableWithTheValue</i>	<i>Define the variable <variable></i>	Given/When/Then I define the variable “<variable>” with the value “<value>”	-	-	-
<i>obtainTheValueFromTheField</i>	<i>Obtain the value from the field <fieldname></i>	Given/When/Then I obtain the value from the field “<fieldname>”	Element	UI Element	Element
Debug Behaviors					
Ontological Behavior	Task	Step of Scenario	UI Elements		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>printOnTheConsoleTheValueOfTheVariable</i>	<i>Print on the console the value of the variable <variable></i>	When/Then I print on the console the value of the variable <variable>	-	-	-
Dialog Behaviors					
Ontological Behavior	Task	Step of Scenario	UI Elements		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>confirmTheDialogBox</i>	<i>Confirm the dialog box</i>	Given/When/Then I confirm the dialog box	Window Dialog	Alert	Dialog
<i>cancelTheDialogBox</i>	<i>Cancel the dialog box</i>	Given/When/Then I cancel the dialog box	Window Dialog	Alert	Dialog
<i>informTheValueInTheDialogBox</i>	<i>Inform the value in the dialog box</i>	Given/When/Then I inform the value “<value>” in the dialog box	Window Dialog	Alert	Dialog

Appendix A: Concept Mapping Table

<i>willBeDisplayedInTheDialogBox</i>	<i>Display <message> in the dialog box</i>	<i>Then will be displayed “<message>” in the dialog box</i>	Window Dialog	Alert	Dialog
Mouse Control Behaviors					
Ontological Behavior	Task	Step of Scenario	UI Elements		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>moveTheMouseOver</i>	<i>Move the mouse over <element></i>	<i>When I move the mouse over “<element>”</i>	Menu	MenuBar	Menu
			MenuItem	Accordion	MenuItem
			Button	Button	Button
			Link	Link	Link
Table Behaviors					
Ontological Behavior	Task	Step of Scenario	UI Elements		
			Ontology	Balsamiq Prototype (com.balsamiq.mockups::)	Final UI
<i>clickOnTheRowOfTheTableReferringTo</i>	<i>Click on the row of the table <table></i>	<i>When/Then I click on the row “<row>” of the table “<table>” referring to “<element>”</i>	Grid	DataGridView	Grid
<i>storeTheCellOfTheTableIn</i>	<i>Store the cell of the table <table> in <place></i>	<i>When/Then I store the cell “<cell>” of the table “<table>” in “<place>”</i>	Grid	DataGridView	Grid
<i>storeTheColumnOfTheTableIn</i>	<i>Store the column of the table <table> in <place></i>	<i>When/Then I store the column “<column>” of the table “<table>” in “<place>”</i>	Grid	DataGridView	Grid
<i>compareTheTextOfTheTableCellWith</i>	<i>Compare the text of the table cell with <text></i>	<i>When/Then I compare the text of the table cell “<table text>” with “<text>”</i>	Grid	DataGridView	Grid
<i>compareTheTextOfTheTableColumnWith</i>	<i>Compare the text of the table column with <text></i>	<i>When/Then I compare the text of the table column “<table text>” with “<text>”</i>	Grid	DataGridView	Grid
<i>clickOnTheCellOfTheTable</i>	<i>Click on the cell of the table <table></i>	<i>When/Then I click on the cell “<cell>” of the table “<table>”</i>	Grid	DataGridView	Grid
<i>clickOnTheColumnOfTheTable</i>	<i>Click on the column of the table <table></i>	<i>When/Then I click on the column “<column>” of the table “<table>”</i>	Grid	DataGridView	Grid

Appendix A: Concept Mapping Table

<i>chooseTheOptionInTheCellOfTheTable</i>	<i>Choose the option in the cell of the table <table></i>	When/Then I choose the option “<option>” in the cell of the table “<table>”	Grid	DataGridView	Grid
<i>chooseTheOptionInTheColumnOfTheTable</i>	<i>Choose the option in the column of the table <table></i>	When/Then I choose the option “<option>” in the column of the table “<table>”	Grid	DataGridView	Grid
<i>typeTheTextInTheCellOfTheTable</i>	<i>Type the text in the cell of the table <table></i>	When/Then I type the text “<text>” in the cell of the table “<table>”	Grid	DataGridView	Grid
<i>typeTheTextInTheColumnOfTheTable</i>	<i>Type the text in the column of the table <table></i>	When/Then I type the text “<text>” in the column of the table “<table>”	Grid	DataGridView	Grid

Appendix B: Log of Results – Assessing Task Models

```
Running story stories/Confirm Flight Selection.storyConverted
User Story: Confirm Flight Selection
(stories/Confirm Flight Selection.storyConverted)

Narrative:
As a IRIT researcher
I want to get all the required data to confirm my flights
So that I can check the information, the fare rules and then finalize my booking.
Scenario: Confirm a Flight Selection
Proceed to Login
Reach the Travel Planet Search Page
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Go to Flight Search - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Go to Flight Search - Task not found! >>
Given I go to "Flight Search"
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Select Round Trip - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Select Round Trip - Task not found! >>
When I select "Round Trip"
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Inform Departure - Task not found! >>
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Choose Departure - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Choose Departure - Task not found! >>
And I inform "Toulouse" and choose "Toulouse, Blagnac (TLS)" in the field "Departure"
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Inform Destination - Task not found! >>
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Choose Destination - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Choose Destination - Task not found! >>
When I inform "Paris" and choose "Paris, Charles-de-Gaulle (CDG)" in the field "Destination"
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Set Departure Date - Found in Position: 8 - Associated Value: Sam, Déc 1, 2018 >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Set Departure Date - Found in Position: 8 - Associated Value: No Value >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Set Departure Date - Found in Position: 15 - Associated Value: No Value >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Set Departure Date - Found in Position: 8 - Associated Value: No Value >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Set Departure Date - Found in Position: 16 - Associated Value: No Value >>
```

Appendix B: Log of Results – Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results – Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results – Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results – Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results – Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results – Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

Appendix B: Log of Results - Assessing Task Models

```
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Choose J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s). - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Choose J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s). - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Choose J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s). - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Choose J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s). - Task not found! >>
When I choose "J'accepte les Conditions d'achat concernant le(s) tarif(s) aérien(s)."
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Click on Decline the trip - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Click on Decline the trip - Task not found! >>
And I click on "Decline the trip"
<< Scenario: No Optional Return Trip With Data.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: No Optional Successful Multidestination Trip - Regular Case.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: Successful Return Trip - Regular Case.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: Successful Multidestination Trip - Regular Case.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: No Optional One-Way Trip Declined.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: Return Trip With Data.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: No Optional Successful Return Trip - Regular Case.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: Successful One-Way Trip - Regular Case.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: One-Way Trip Declined.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
<< Scenario: No Optional Successful One-Way Trip - Regular Case.scen - Searched Task: Display Votre voyage a été annulé! - Task not found! >>
Then will be displayed "Votre voyage a été annulé!"
```

Annex A: Case Study Interview Protocol

Bonjour !

Tout d'abord, merci d'avoir accepté cet entretien.

Je m'appelle Thiago Silva et je suis doctorant dans l'équipe ICS de l'IRIT.

Dans le cadre de ma thèse de doctorat j'étudie des techniques pour spécifier les besoins des utilisateurs et puis les tester sur le logiciel.

Aujourd'hui je souhaite récupérer des informations pour réaliser une étude de cas sur le traitement des voyages d'affaires. A ce titre, je vous propose un entretien d'environ une heure au cours duquel vous serez invité à répondre à des questions sur différents aspects du processus de réservation de voyages d'affaires pour les membres de l'institut.

L'objectif principal de cet entretien est de récupérer des informations qu'on pourra spécifier en tant que **besoin de voyageurs** sur un format qu'on appelle « **récit utilisateur/user story** ». Une fois que je vous aurais expliqué ce qu'on entend par « récit utilisateur/user story » je vous demanderais de me faire parvenir quelques exemples de récits utilisateurs que vous auriez entendu/reçu cette semaine.

Avec votre accord, cet entretien sera enregistré. Néanmoins je vous rassure que tout ce que vous dites restera anonyme et confidentielle.

Si vous avez des questions ou des doutes sur l'entretien ou sur les questions qui seront posées, n'hésitez pas à nous interrompre et à demander plus d'informations.

Faire signer consentement éclairé.

Initier l'enregistrement.

Partie I : Questionnaire démographique et de contexte

Question 1. Pourriez-vous répondre à l'ensemble des informations de base ci-dessous ?

A. Votre sexe :

B. Votre âge :

C. Votre niveau d'étude :

D. Depuis combien de temps vous êtes au service de mission de l'IRIT ?

E. Avez-vous déjà eu des expériences dans de services similaires auparavant ?

Question 2. Pourriez-vous nous donner un aperçu de ce travail, en fournissant une brève description de vos tâches ?

Partie II : Processus de traitement de demandes

Nous sommes intéressés par les préférences et les difficultés que les voyageurs de l'IRIT ont rencontrées et vous ont signalées lorsqu'ils essaient de réserver leurs voyages d'affaires. Nous sommes également intéressés par votre opinion sur les demandes reçues.

Label : * faits, • interprétation

Section A : Réception de demandes de réservation

Question 1. Comment les demandes de réservation des voyageurs arrivent-elles à vous et avec quelle fréquence ? Avez-vous des suggestions pour faire mieux ? *

Annex A: Case Study Interview Protocol

Question 2. Combien de demandes de réservation de voyage avez-vous reçues la semaine dernière ? Avez-vous des suggestions pour faire mieux ? *

Question 3. Pensez-vous qu'il manque quelque chose dans la description des demandes de réservation que vous recevez ? Comment cela pourrait-il être mieux ? •

Question 4. Est-ce que vous devez prendre des notes (ex. post-it, email, etc.) sur les demandes de réservations ? Si oui, combien de notes en moyenne ? Comment cela pourrait-il être mieux ? Combien de notes avez-vous pris la semaine dernière ? *

Question 5. Si vous prenez des notes, comment vous les conservée et sur quel format ? Comment améliorer l'enregistrement de ces notes ? *

Question 6. Pensez-vous que l'enregistrement de ces notes est important ? Comment cela pourrait-il être mieux ? •

Question 7. Pouvez-vous fournir quelques exemples de demandes de réservation que vous recevez ? Comment cela pourrait-il être mieux ? *

Section B : Traitement des demandes de réservation

Question 8. Quelle est la procédure type pour traiter une demande de réservation ? Comment cela pourrait-il être mieux ? *

Question 9. Est-ce que dans les demandes des voyageurs que vous traitez il y a des informations qu'on pourrait identifier comme des besoins et/ou des exigences pour améliorer un système de réservation de voyage ? Si oui, pouvez-vous les quantifier et les identifier ? •

Annex A: Case Study Interview Protocol

Question 10. Quelles sont vos besoins/exigences pour le système de réservation que vous utilisez actuellement ? •

Question 11. Avec quelle fréquence vous devez demander l'aide des autres membres de l'équipe pour résoudre les demandes des voyageurs ? Comment cela pourrait-il être mieux ? *

Question 12. Avec quelle fréquence vous devez demander aux voyageurs de clarifier les informations concernant leur demande ? Comment cela pourrait-il être mieux ? *

Question 13. Selon votre propre expérience, quelles seraient les fonctionnalités qui vous seraient utiles et qui devraient être rajoutées au logiciel de réservation ? •

Annex A: Case Study Interview Protocol

Question 14. Selon votre propre expérience, quelles seraient les fonctionnalités qui seraient utile pour les voyageurs ? •

Question 15. Pourriez-vous lister 3 fonctionnalités que vous aimerais garder pour ce type de système ? •

Question 16. Pourriez-vous lister 3 fonctionnalités que vous aimerais changer pour ce type de système ? •

Section C : Rédaction des Récits Utilisateurs

Veillez trouver ci-joint un exemple de spécification d'un « **récit utilisateur/user story** ».

Présentez et expliquez le modèle des Récits Utilisateur :

Titre (Une ligne décrivant l'histoire)

Préambule:

En tant que [rôle ou personne]
Je veux [fonctionnalité]
Afin de [but, bénéfice ou valeur de la fonctionnalité]

Scénario 1: [description]

Etant donné [un contexte initial (les acquis)]
Et [un autre contexte]...
Quand [un événement survient]
Alors [on s'assure de l'obtention de certains résultats]
Et [un autre résultat]...

Scénario 2: ...

Un exemple :

Titre: Recherche de billets d'avion

Préambule:

En tant que voyageur fréquent,
Je veux rechercher des billets, en fournissant des emplacements et des dates,
Afin de pouvoir obtenir des informations sur les tarifs et les horaires des vols.

Scénario: Recherche de tickets "aller simple"

Etant donné que je vais à la page "Recherche de vols"
Quand je choisis "aller simple"
Et je tape "Paris" et choisis "Paris, Charles de Gaulle (CDG)" dans le champ "Départ de"
Et je tape "Toulouse" et choisis "Toulouse, Blagnac (TLS)" dans le champ "Arrivée à"
Et je choisis "2" dans le champ "Nombre total de passagers"
Et je choisis "15/12/2017" dans le champ "Date de départ"
Et je clique sur "Rechercher"
Alors il sera affiché la liste des vols disponibles

Question 1. Est-ce que vous pourriez spécifier un exemple des demandes de réservation en utilisant ce format de description ? •

- Pensez-vous pouvoir rédiger une liste de demandes/problèmes que vous recevrez au cours de cette semaine sur les problèmes rencontrés par les utilisateurs lors de la réservation de leurs voyages d'affaires ?

Annex A: Case Study Interview Protocol

- Si oui, pensez-vous que vous pouvez formater ces demandes/problèmes en suivant le modèle « récits utilisateur/user stories » que j'ai présenté tout à l'heure ?

L'entretien est maintenant fini, merci beaucoup de votre participation !

Annex B: User Stories Written by the Case Study Participants

Preamble .

En tant que : Invité .

J'aimerais : Des billets d'avion avec horaires et vols définis .

Alors je : Réserve des billets .

Scénario : Recherche du billet demandé .

Etant donné : Je suis sur le site . (SIMBABA/TRAVEL)

Quand je choisis le vol demandé

ENsuite
Et : Je choisi type de voyageur - "Invités"
Toulouse / PARIS .

ET : Départ 7H00 : Recherche même pour 10h00 .
ALORS : RECHERCHE → Plusieurs propositions .
Et : JE choisi le vol . Derniers .
On choisit le voyageur .

— Quand . On Je sais les données concernant

le voyageur - (nom , prénom , date naissance ,
téléphone , email) ;
eventuellement les cartes à Réductions .
(Flying blue et Carte abonnement) .

ALORS - Billet en demande de validation .

En tant que voyageur fréquent,

Je veux rechercher des billets, en fournissant des emplacements et des dates pour un voyage multi destinations.

Afin de pouvoir obtenir des informations sur les tarifs et les horaires des vols.

Scénario : Recherche d'un billet multi destinations

Etant donné que je vais à la page "recherche de vols"

Et quand je choisis "multi-destinations"

Et je tape "Paris" et choisis "Paris, Charles de Gaulle, dans le champ "départ de"

Et je choisis "Rio de Janeiro" dans le champs "arrivée à"

Et je choisis 15/02/17 dans le champ "date de départ"

Et je choisis 20/02/17 dans le champ "date de retour"

Et je tape Rio de Janeiro dans le champs "arrivee à" de "

Et je tape Porto Alegre dans le champs "arrivée à"

Et je choisis 17/02/17 dans le champs "date de départ"

Et je choisis 19/02/17 dans le champs "date d'arrivée"

Et je clique sur "rechercher"

Alors il sera affiché la liste des vols disponibles

- En tant que : gestionnaire

- je veux : consulter les autorisations de voyage
afin de m'assurer des réservations effectives.

Scénario : liste autorisation de voyage

Etant donné : je veux voir l'onglet autorisation de voyage
quand : je tape la référence de réservation
et : je consulte la demande si elle a bien été
enregistrée.

Alors : A ce moment-là, je peux être sûr ou pas
que la réservation a été prise en compte.

- Apparition d'un onglet : Autorisation n'autorisé

En tant que Stagiaire

Je veux réserver un vol en direction de Paris pour
un départ le 2 Mai jusqu'au 10 Mai
Afin de participer à un séminaire.

Scénario :

Etant donné que je vais réserver mon vol
Quand je choisis la case recherche renseigne
toutes les infos.

Et que je choisis recherche par tarif
Alors tous les vols disponibles la journée sont
classés par prix croissant.

Annex C: Transcription of the Interviews

1. TRANSCRIPTION : Participant 1 (P1)

Partie I : Questionnaire démographique et de contexte.

Interviewer : Bon, voilà ! La première partie, c'est un bref questionnaire démographique. Votre sexe ? Votre âge ?

P1 : Féminin, Cinquante.

Interviewer : Niveau d'étude ?

P1 : Bac + 1

Interviewer : Depuis combien de temps, vous êtes au service de mission de l'IRIT ?

P1 : Quatre ans

Interviewer : Avez-vous déjà eu des expériences dans de services similaires auparavant ?

P1 : Oui

Interviewer : Combien du temps ?

P1 : Six ans.

Interviewer : Pourriez-vous nous donner un aperçu de ce travail, de vos tâches spécifiquement ?

P1 : De tout ?

Interviewer : Non, non ! La partie de la réservation de voyage.

P1 : De réservation de voyage ?

Interviewer : Oui.

P1 : En ce qui me concerne maintenant les agents de laboratoire, ils ont fait la réservation sur le site. Et ensuite pour les invités, on fait que le bon de commande et le valide le billet et en c'est qui concerne les invités on va faire, nous-même la réservation.

Interviewer : Donc, parfois c'est les chercheurs que fassent la programmation de voyage et parfois c'est vous-même ?

P1 : C'est ça.

Interviewer : D'accord ! Et ça dépend de quoi ?

P1 : Alors... c'est que nous... on fait pour les invités.

Interviewer : D'accord !

P1 : Les invités, les stagiaires qui n'ont pas l'accès à l'intranet.

Interviewer : D'accord ! Et si les chercheurs n'arrivent pas de faire tout seul ?

P1 : On les aide. D'abord, on fait ensemble. S'ils n'arrivent pas on peut faire la demande directement par mail.

Interviewer : D'accord !

Partie II : Processus de traitement de demandes

Nous sommes intéressés par les préférences et les difficultés que les voyageurs de l'IRIT ont rencontrées et vous ont signalés lorsqu'ils essaient de réserver leurs voyages d'affaires. Nous sommes également intéressés par votre opinion sur les demandes reçues.

Label : * faits, • interprétation

Section A : Réception de demandes de réservation

Interviewer : Concernant dans le traitement de demande. D'abord la réception de demande. Comment les demandes de réservation des voyageurs arrivent-elles à vous et avec quelle fréquence ? *

P1 : La fréquence c'est compliquée.

Interviewer : En moyenne ?

P1 : 400 missions par an.

Interviewer : Par an ?

P1 : Par an ! Oui !

Interviewer : 400 missions par an ? Pour chacun ?

P1 : Pour moi. Moi, je traite 400 demandes de mission par an. Comment elles arrivent ? Je ne sais pas si vous connaissez le GLPI ou le « *Travel planet* » ?

Interviewer : Pas trop.

P1 : Voilà, les demandes d'ordre de mission sont dans le GLPI et parallèlement les chercheurs font leurs demandes de réservation sur le site.

Interviewer : D'accord !

P1 : Donc, on reçoit d'un côté la demande d'ordre de mission et ça sera la justificative de la mission. Et parallèlement on a le billet de la demande de réservation que nous arrive par mail.

Interviewer : D'accord ! Donc, vous utilisez deux systèmes. Le GLPI et le système de réservation ?

P1 : Le GLPI c'est pour le labo ; pour la réservation du billet on est obligé de toute façon d'aller sur...souvent si la mission est sur l'université ou CNRS, on est obligé d'aller sur la plateforme, donc, le marché. On doit aller sur la plateforme.

Interviewer : D'accord ! Et s'appelle comment cette plateforme ?

P1 : Pour l'université s'appelle « *Travel Planet* » pour le CNRS... je n'ai pas dans la tête.

Interviewer : D'accord !

P1 : Bon, s'appelle SIMBAD.

Interviewer : D'accord ! Et c'est quel l'ordre des choses ? D'abord on va au GLPI et après on vas au « *Travel Planet* » par exemple ?

P1 : Il fait comme ils veulent. L'importance c'est quand on... (audio inaudible), nous, on va tout voir sur le GLPI, la demande d'ordre de mission, parce qu'on ne peut pas les aider si on ne voit pas les éléments nécessaires.

Interviewer : D'accord !

P1 : Voilà... (audio inaudible).

Interviewer : D'accord ! Dans le côté chercheur peu importe quel ordre il fait les choses.

P1 : Soit il fait parallèlement... (audio inaudible).

Interviewer : D'accord ! Combien de demandes de réservation de voyage avez-vous reçues la semaine dernière ? Quelques idées ? Après prêt ? *

P1 : Une dizaine de demande.

Interviewer : Une dizaine dans la semaine dernière ?

P1 : Oui !

Interviewer : D'accord Concernant l'arrivée de demande, avez-vous de suggestion ? *

P1 : Sincèrement, ça dépend.

Interviewer : Oui, oui. C'est en moyenne.

P1 : En fin d'année, par contre, voilà c'est douze.

Interviewer : Douze par jour ?

P1 : Ça arrive !

Interviewer : D'accord ! En moyenne c'est douze par semaine ?

P1 : À la semaine dernière oui. Mais, voilà, il y a de semaine que c'est deux ou trois demandes. Mais, il y a d'autres qu'on peut voir plus.

Interviewer : D'accord ! Concernant l'arrivée de demande avez-vous de suggestion pour faire mieux ce processus, ou il est déjà bon ?

P1 : Bah ! Voilà ! (Audio inaudible).

Interviewer : D'accord ! C'est bon comme ça.

P1 : C'est très bien.

Interviewer : Mais vous pensez qu'il y a de problème pour faire ça, dans deux systèmes différents ?

P1 : Non !

Interviewer : Ça ne pose pas de souci ?

P1 : Non, parce que le portail de réservation de billets, ils sont... donc, on peut faire en parallèle.

Interviewer : D'accord ! Pensez-vous qu'il manque quelque chose dans la description des demandes de réservation que vous recevez ?

P1 : Dans la description ?

Interviewer : La description de réservation de voyage.

P1 : Dans la demande d'ordre de mission qu'on a de problème dans le voyage...

Interviewer : Le voyage, la demande de réservation de billet d'avion.

P1 : C'est pareil. Tout dépend comme est la formule. Quand le chercheur a une disponibilité, on fait la demande... (audio inaudible).

Interviewer : D'accord ! Mais en moyenne, ça arrive assez complet la description ?

P1 : Pas trop.

Interviewer : D'accord.

P1 : Ça dépend du chercheur.

Interviewer : D'accord ! Comment cela pourrait-il c'est mieux ? La description de demande ? Vous avez de suggestion ?

P1 : Voilà. C'est ça. Si le chercheur avait de disponibilité des horaires, donner les informations, ça nous facilite le travail. Voilà !

Interviewer : D'accord ! Est-ce que vous devez prendre des notes, par exemple : un post-it, mail, quelque note, etc., sur les demandes de réservations ? Ou non, c'est assez complet ? *

P1 : Non, sincèrement, si on a besoin, on va relier, parce que le GLPI est, peut-être, notre post-it.

Interviewer : D'accord

P1 : Parce qu'il y a un suivi dans le GLPI. Car nous manque quelque chose, on peut demander au-dessous et ils répondent à la fin de compte. Voilà, c'est tous les éléments sur les autres.

Interviewer : D'accord !

P1 : Après, évidemment on note de choses pour clarifier les choses... (audio inaudible).

Interviewer : D'accord ! Donc, bon, ça arrive. Combien de notes en moyenne ? et avec quelle fréquence ? Vous avez d'idée ? *

P1 : Ça dépend. Pour le billet du train c'est plus que par le billet d'avion. (Audio inaudible).

Interviewer : D'accord ! Quelque suggestion pour faire mieux, pour améliorer c'est processus de prendre de notes ? Dans le système ou dehors le système ?

P1 : Améliorer c'est que la demande soit plus claire et précise au départ. Si la demande au départ elle est bien faite, normalement, nous, on n'a pas de problème. C'est super-facile.

Interviewer : D'accord ! Mais vous croyez que ça c'est n'est pas un problème du système en fait.

P1 : Je ne pense pas.

Interviewer : Non ?

P1 : Non !

Interviewer : Pour les utilisateurs, les chercheurs, par exemple. S'ils ne suivent pas forcément la procédure pour la réservation, vous ne pensez pas que c'est à cause d'un problème du système ?

P1 : Après, je pense qu'au départ, je ne généralise pas. Les demandes sont, ou peut-être, souvent ... (audio inaudible).

Interviewer : D'accord ! Si vous prenez des notes, comment vous les conservées, c'est le cas. Vous avez me dit à tout à l'heure c'est au GLPI ? *

P1 : GLPI

Interviewer : Il n'y a pas de note dehors le système ? Normalement ?

P1 : Si, on est tous pareil, un mettre un petit rappelle sur le dossier.

Interviewer : Mais vous enregistrer sur le système après ?

P1 : Oui !

Interviewer : Ou non ?

P1 : Ils sont tous dans le système.

Interviewer : D'accord.

P1 : Tout est dedans.

Interviewer : Bon, comment améliorer ce processus-là ?

Je crois que vous avez dit à tout à l'heure, c'est avoir une demande claire et précise au départ, c'est ça ?

P1 : Oui !

Interviewer : Pensez-vous que l'enregistrement de ces notes est important ? •

P1 : Oui !

Interviewer : Oui ?

P1 : Oui, parce que le dossier suivi sur le GLPI... (audio inaudible).

Interviewer : D'accord. Il y a quelque suggestion ? Sur l'enregistrement de ces notes supplémentaires ?

P1 : Non !

Interviewer : Non ?

P1 : Non ! C'est bon comme ça !

Interviewer : Donc, pour l'instant pas de suggestion ?

P1 : Non.

Interviewer : Pouvez-vous fournir quelques exemples de demande de réservation que vous recevez ? Dans quel format ? *

P1 : Vous voulez un exemple de demande ?

Interviewer : Oui !

P1 : En fait ?

Interviewer : Oui !

P1 : (Audio inaudible).

Interviewer : D'accord !

P1 : Dans la demande il y a tout qu'on a besoin. Il y a de compte, il y a les dates, il y a...

Interviewer : Donc, s'il y a tout rempli...

P1 : Dans l'ordre de mission on a tous les infos nécessaires pour faire la mission. Après on ajoute les justificatifs et voilà.

Interviewer : D'accord !

P1 : (Audio inaudible).

Interviewer : D'accord ! Donc, cette demande, elle arrive par mail, c'est ça ? Avec tous les éléments ?

P1 : GLPI

Interviewer : Oui, mais le GLPI, Il envoie le mail, avec...

P1 : Non.

Interviewer : Non ? Il faut que vous connectiez ?

P1 : Et voilà !

Interviewer : D'accord !

P1 : (Audio inaudible).

Interviewer : D'accord ! Mais arrive d'un chercheur faire la demande directement par mail ? Sans passer par le GLPI ?

P1 : Oui !

Interviewer : Oui ? Ça arrive ?

P1 : Oui !

Interviewer : D'accord !

Section B : Traitement des demandes de réservation.

Interviewer : Concernant le traitement des demandes. Quelle est la procédure-type de traiter une demande de réservation ? Comment ça arrive ? Tout d'abord on fait ça, après on fait ça. C'est quel le processus ? *

P1 : Ça c'est la mission en général. Donc, pour la demande de validation payée, on fait de bon de commande, ensuite, on va retourner sur le portail pour mettre le bon de commande. Voilà !

Interviewer : D'accord. Donc, c'est la demande, le bon de commande et le portail pour mettre le bon de commande pour valider. C'est dans le portail qu'on valide ?

P1 : Oui !

Interviewer : Le portail ce n'est pas le « *travel* ».

P1 : Le portail c'est le SIMBAD.

Interviewer : Portail et « *travel planet* » c'est la même chose ?

P1 : Il y a deux systèmes différents. (Audio inaudible).

Interviewer : D'accord ! Est-ce que dans les demandes des voyageurs, que vous traitez, il y a des informations qu'on pourrait identifier comme des besoins et/ou des exigences pour améliorer un système de réservation de voyage ? •

P1 : Un problème c'est quand on a des voyages multiples-destinations. C'est un peu compliqué à gérer.

Interviewer : Pourquoi ?

P1 : Au niveau de la réservation le portail et le logiciel sont compliqués les multiples-destinations. Parce qu'on n'arrive pas à ajouter une ville, même s'il a in départ Toulouse, mais on part de Brive, d'autre ville... (audio inaudible).

Interviewer : D'accord !

P1 : Ça c'est l'agence de voyages.

Interviewer : Mais vous croyez que c'est un problème de logiciel ?

P1 : Certains pays avec le multiple-destination, c'est un peu compliqué.

Interviewer : De train et d'avion aussi ?

P1 : Sur tout de train, parce que de train on n'arrive pas de le faire. C'est un peu compliqué pour récupérer de billet. On ne peut pas toujours faire de réservation de billet de train à l'Allemagne ou à l'Italie.

Interviewer : D'accord !

P1 : (Audio inaudible).

Interviewer : D'accord. Quelles sont vos besoins/exigences pour le système de réservation de voyage que vous utilisez actuellement ? •

Bon ! Donc, j'imagine que c'est gérer les multiples- destinations. C'est ça ?

P1 : Oui ! Voilà !

Interviewer : Il y a quelque d'autre ?

P1 : Quelque chose d'autre ?

Interviewer : Oui !

P1 : Voyage en train à l'étranger, que c'est un peu compliqué, mais... C'est tout !

Interviewer : D'accord ! Avec quelle fréquence vous devez demander l'aide des autres membres de l'équipe pour résoudre les demandes des voyageurs ? *

P1 : Rarement !

Interviewer : Rarement ?

P1 : Rarement vraiment.

Interviewer : Est-ce que vous avez de suggestion pour améliorer cet échange d'information entre l'équipe ?

P1 : Entre nous ?

Interviewer : Oui !

P1 : Non ! entre nous non. Parce qu'on est dans un « *open space* » donc, ça marche bien les échanges.

Interviewer : D'accord ! Donc, ça marche bien ?! Avec quelle fréquence vous devez demander aux voyageurs de clarifier les informations concernant leur demande ? *

P1 : Plutôt souvent.

Interviewer : Plutôt souvent ?

P1 : Non, non ! Ce n'est pas ! Parce qu'on n'a pas de soucis. Dans le dossier, c'est juste un petit « *delay* ». Donc, c'est rare quand il y a tout complet. (Audio inaudible).

Interviewer : D'accord.

P1 : (Audio inaudible).

Interviewer : D'accord ! Avez-vous des suggestions pour faire ça mieux ? La clarification des choses avec les voyageurs. Voie système, je ne sais pas.

P1 : Après c'est qu'ils soient plus précis possible. Voilà. Parce que, nous, on a besoin de précision au niveau d'horaire et tout ça. C'est plus facile pour nous.

Interviewer : D'accord ! Selon votre propre expérience, quelles seraient les fonctionnalités qui vous seraient utiles et qui devraient être rajoutées au logiciel de réservation ? •

P1 : Ce n'est pas trop. Parce qu'au niveau des tarifs il nous propose le moins cher aussi !

Interviewer : Ça n'existe pas ?

P1 : Ah ! La date de validation !

Interviewer : Comment ?

P1 : La date de validation est après le devis. C'est la date qu'on doit valider. Notre problème c'est qu'à ce moment-là, le « *delay* » est court... (audio inaudible).

Interviewer : Donc, le problème dans c'est cas-là c'est que le « *delay* » est trop court.

P1 : On perd le billet entre le devis et le moment de validation. Principalement pendant le week-end. (Audio inaudible).

Interviewer : D'accord ! Bon ! Quelles seraient les fonctionnalités qui seraient utile pour les voyageurs ? À votre avis ? •

P1 : Je ne sais pas. Parce que pour moi, la réservation d'un vol c'est pareil.

Interviewer : Donc, il n'y a pas de choses que pourrait améliorer à ce niveau-là.

P1 : Non. (Audio inaudible).

Interviewer : D'accord ! Pourriez-vous lister 3 fonctionnalités que vous aimeriez garder pour ce type de système ? Le système de voyage, pas le GLPI. •

P1 : Garder ?

Interviewer : Oui ! Que vous aimez plus ou que vous considérez essentiel.

P1 : Les différents propositions au niveau des horaires, des tarifs. (Audio inaudible).

Interviewer : D'accord ! Pourriez-vous lister trois fonctionnalités que vous aimeriez changer pour ce type de système ? .

P1 : Changer ou complètement changer ?

Interviewer : Oui !

Interviewer : Après le système est bien présenté. Pour les multiples-destinations, il ne marche pas toujours. Et le changement qu'on peut voir c'est quand il y a plusieurs voyageurs qui partent au même temps, et au même endroit. On doit faire des réservations différentes.

Section C : Rédaction des Récits Utilisateurs.

Interviewer : Bon ! Dans cette dernière partie, c'est le modèle qu'on va essayer de tester avec vous.

Comme j'ai dit à tout à l'heure, c'est un modèle pour décrire une fonctionnalité, décrire une demande, D'accord ? Donc, il est toujours comme ça. Il a un titre pour décrire le type d'histoire, on appelle ce modèle, de récit utilisateur. Donc on a un préambule.

En tant que [rôle ou personne]

Je veux [fonctionnalité]

Afin de [but, bénéfice ou valeur de la fonctionnalité]

Scenario 1 : [description]

Étant donné [un contexte initial (les acquis)]

Et [un autre contexte] ...

Quand [un évènement survient]

Alors [on s'assure de l'obtention de certains résultats]

Et [un autre résultat] ...

Un exemple :

En tant donné que je vais à la page "Recherche des vols"

Quand je choisis : "aller simple"

Et je tape "Paris" et choisis "Paris, Charles de Gaulle (CDG)" dans le champ "Départ de"

Et je tape "Toulouse" et choisis "Toulouse, Blagnac (TLS)" dans le champ "Arrivée à"

Et je choisis "2" dans le champ "Nombre total de passagers"

Et je choisis "15/12/2017" dans le champ "Date de départ"

Et je clique sur "Recherche"

Alors le système va afficher la liste des vols disponibles.

Donc, c'est un récit en fait, un modèle pour décrire la demande.

P1 : Mais c'est individuelle la demande.

Interviewer : Oui, mais ça ce n'est pas fixe, c'est un exemple général. On n'est pas obligé de décrire les choses comme ça. On est obligé d'utiliser ces éléments-là. Pour avoir une histoire on est obligé d'avoir : **Étant que, Je veux, Afin de,** et pour chaque scénario on est obligé d'avoir : **Étant Donnée, Quand, Et, Alors.** Mais, quoi on met dedans n'importe. Il faut décrire le processus d'une demande avec ce modèle. Donc, c'est ça qu'on va évaluer. Si c'est modèle est bien adapté, s'il est facile d'utiliser ou pas. Vous pensez que c'est possible d'écrire une demande en suivant ce modèle-là ?

P1 : Oui, on va faire.

Interviewer : D'accord ! Quel type de demande ?

P1 : Invité.

Interviewer : D'accord ! Avec un voyage en train ou même un voyage multiples-destinations. On n'est pas obligé de faire la tâche recherche du vol.

P1 : (Elle écrit et parle en voix bas)

Interviewer : Vous avez besoin donner un contexte. Quand quelque chose arrive...

P1 : (Elle écrit et parle en voix bas)

Interviewer : Donc, si c'est un invité, vous n'êtes pas obligé de choisir l'utilisateur ?

P1 : Non. C'est après qu'on a fait le billet. (Elle écrit et parle en voix bas).

Interviewer : D'accord !

P1 : (Elle écrit et parle en voix bas).

Interviewer : Vous devez remplir ça après avoir soumis ?

P1 : Oui !

Interviewer : Donc, à ce modèle-là, si vous faites...

P1 : Je choisis... (Elle écrit et parle en voix bas).

Interviewer : D'accord ! Il manque juste une conclusion !

P1 : (Elle écrit en lisant quelque chose en voix bas).

Interviewer : Quand il fait la recherche c'est quoi qu'arrive ? C'est quelle conséquence ? Ça c'est important de dire.

P1 : (Elle écrit)

Interviewer : D'accord ! Très bien ! C'est bon comme ça ! Donc, pensez-vous pouvoir rédiger une liste de demande/ problème, surtout de problème que vous recevrez au cours cette semaine et de la semaine prochaine ? Par exemple, que les voyageurs vont vous demander ? Par exemple. Je n'arrive pas à chercher de vol moins cher.

P1 : Oui.

Interviewer : Donc, vous pouvez faire une liste de problèmes que vous recevez. Si oui, bon, c'est le cas, pensez-vous pouvoir formater ces demandes/ problèmes dans ce modèle-là ? C'est juste que vous avez fait.

P1 : Oui !

Interviewer : Donc, c'est ça que je vous demande. Une liste de demande/ problème. Dans le cadre d'une semaine. Aujourd'hui c'est mardi, jusqu'à mardi prochain, par exemple. Et après vous écrivez. Vous pensez que c'est beaucoup de problèmes ?

P1 : Non.

Interviewer : Non ?

P1 : On ne sait pas.

Interviewer : On est attaché juste à la recherche du vol, de train, les autres problèmes liés au système, vous pouvez laisser à côté. Donc, je vous demande d'envoyer par courriel électronique les erreurs et les demandes formatées. D'accord ?

P1 : Oui !

Interviewer : Et après, je vais vous envoyer un petit questionnaire pour que vous puisez évaluer ce type de format-là.

P1 : D'accord !

Interviewer : Donc, voilà ! Merci beaucoup !

2. TRANSCRIPTION : Participant 2 (P2)

Partie I : Questionnaire démographique et de contexte.

Interviewer : Donc, la première partie, concerne un questionnaire démographique. D'accord ?

P2 : D'accord !

Interviewer : Donc, votre sexe ?

P2 : Féminin. On ne sait jamais.

Interviewer : Il faut demander. Votre âge ?

P2 : 30 ans

Interviewer : Votre niveau d'étude ?

P2 : Bac + 2

Interviewer : Depuis combien de temps, vous êtes au service de mission de l'IRIT ?

P2 : Trois ans

Interviewer : Avez-vous déjà eu des expériences dans des services similaires ? D'autre part ?

P2 : Oui

Interviewer : Combien du temps ?

P2 : Avant le IRIT, trois ans.

Interviewer : Donc. Vous avez six ans d'expérience dans ce type de service ?

P2 : Oui.

Interviewer : D'accord ! Pourriez-vous nous donner un aperçu de ce travail, en fournissant une brève description de vos tâches ?

P2 : Description de mes tâches ?

Interviewer : Oui ! D'une manière générale.

P2 : De manière générale, déjà notre mission, c'est de gérer le portefeuille de chaque équipe. J'ai eu septe d'un projet, tous qui est mission, donc de réservation de vols, d'hébergement, ensuite je m'occupe des missions de côté de remboursement et ensuite je m'occupe d'achat, des livrassions, des facturations et, voilà, en gros, voilà, c'est ça.

Interviewer : Donc, le traitement de voyage c'est qu'une petite partie.

P2 : C'est une grande partie.

Interviewer : C'est une grande partie !

P2 : Parce qu'il y a beaucoup de missions, il y a beaucoup d'achat. On doit faire attention au marché et tout ça. Il y a beaucoup de missions effectivement, donc, c'est une grande partie de réservation de transport.

Interviewer : D'accord ! Très bien !

Partie II : Processus de traitement de demandes

Nous sommes intéressés par les préférences et les difficultés que les voyageurs de l'IRIT ont rencontrées et vous ont signalés lorsqu'ils essaient de réserver leurs voyages d'affaires. Nous sommes également intéressés par votre opinion sur les demandes reçues.

Label : * faits, • interprétation

Section A : Réception de demandes de réservation

Interviewer : Bon, concernant sur la réception de demandes. Comment les demandes de réservation des voyageurs arrivent-elles à vous et avec quelle fréquence ? *

P2 : Alors, la demande de réservation on reçoit par mail.

Interviewer : Par mail ?

P2 : Voilà ! Comme vous, je pense que vous recevez aussi par mail. C'est une demande d'accord d'abord. Ensuite, donc, c'est à ce moment-là que nous, on établit un bon de commande, pour réserver l'argent et ensuite, on retourne sur le site et on mentionne le numéro de commande. Comme ça, l'agence comptable possède la facture.

Interviewer : D'accord ! Et avec quelle fréquence ces demandes arrivent ?

P2 : Ça dépend, bien sûr, ça dépend de mois, ça dépend du jour.

Interviewer : En moyenne ?

P2 : En moyenne, il y a, on va dire, trois quatre par jour.

Interviewer : Trois, quatre par jour ?

P2 : Après prêt ! Après, ça dépend.

Interviewer : D'accord. Et avez-vous de suggestion pour améliorer ce processus de réception de demande ?

P2 : Non !

Interviewer : Non ? Vous pensez que c'est bon comme ça ?

P2 : Parce qu'avant de passer, on reçoit le mail, on consulte le GLPI, on appelle la plateforme, donc, là on n'a pas le risque d'erreur.

Interviewer : Donc, tout d'abord ils essaient de faire la réservation et s'ils ont des soucis c'est juste dans ce cas-là qu'ils font de contact avec vous ? Ou non ? Ils font contact dans tous les cas, n'importe pas s'ils ont de problème ou pas ?

P2 : Bon de toute façon ils mettent en place, ils rappellent nous avant de faire quoique ce soit parce qu'ils ne savent pas, c'est-à-dire. Après, sinon, à cause d'un problème ils rappellent nous. Parce qu'après, en parallèle ils font la demande, parce qu'on n'a pas la demande de mission à côté, donc, en parallèle ils nous contactent par le GLPI. On n'a pas encore une relation... (audio inaudible).

Interviewer : D'accord ! Donc il faut que la demande arrive par mail. C'est idéal ? La demande qui vous intervenez.

P2 : Oui, mais d'abord c'est idéal de déposer la demande de mission sur le GLPI et ensuite on reçoit la demande par mail.

Interviewer : Donc, si arrive par téléphone ça pose de problèmes normalement

P2 : Par téléphone... (audio inaudible).

Interviewer : D'accord ! Très bien ! Combien de demandes de réservation de voyage avez-vous reçues la semaine dernière ? Savez-vous ? Après prêt ? *

P2 : Une dizaine.

Interviewer : Une dizaine ?

P2 : Oui !

Interviewer : D'accord ! Bon, avez-vous des suggestions pour faire mieux dans cette réception ? *

P2 : Non !

Interviewer : Avec le volume aussi, bien sûr !

P2 : Avec ?

Interviewer : Le volume de demandes.

P2 : Non, non !

Interviewer : Pensez-vous qu'il manque quelque chose dans la description des demandes de réservation que vous recevez ? *

P2 : La description ?

Interviewer : Oui !

P2 : Non !

Interviewer : Non ? Elles viennent toutes complètes normalement ?

P2 : Oui !

Interviewer : Vous voyez une façon de faire mieux ?

P2 : Non !

Interviewer : C'est bon comme ça ?

P2 : C'est bon comme ça.

Interviewer : Dans la description spécifiquement ?

P2 : Oui !

Interviewer : Est-ce que vous devez prendre des notes, par exemple : un post-it, dehors le système, sur les demandes de réservations que vous recevez ? *

P2 : Non !

Interviewer : Non ? Elles viennent toutes complètes ?

P2 : Oui !

Interviewer : Donc, si oui, combien de notes en moyenne ? Donc, ça ne fait pas de sens. Comment cela pourrait-il être mieux ? Pas de suggestion ? *

P2 : Non, mais... franchement non.

Interviewer : Si vous prenez des notes, comment vous les conservées et sur quel format ? Au cas où ! *

P2 : Si on prend des notes on va prendre un post-it et on va mettre dans une poche avec le dossier.

Interviewer : D'accord ! Donc, vous imprimez ces notes ?

P2 : De toute façon on va imprimer, la demande de réservation, s'il y a de choses à rajouter ou des informations on va noter sur le post-it et avec cette demande de réservation on va mettre dans le dossier.

Interviewer : D'accord.

P2 : On fait tout pour que le dossier soit complet. Quand on a besoin les informations du dossier, il est bien ranger.

Interviewer : Oui ! Comment améliorer ces notes ? Vous-avez des suggestions ?

P2 : Après, si j'avais des altérations à faire aussi, autant que pour les chercheurs que pour nous dans le GLPI et dans ce cas, on se vérifie sur le ticket et là... (audio inaudible).

Interviewer : D'accord.

P2 : Ça aussi c'est sympa. Car on récapitule tous les échanges.

Interviewer : Ça existe déjà ?

P2 : Et voilà !

Interviewer : Donc, pensez-vous que l'enregistrement de ces notes est important ? •

P2 : Ah ! Oui !

Interviewer : Oui ?

P2 : Oui.

Interviewer : Bon, et la façon de faire mieux, c'est même le suivi des notes que vous avez déjà dans le système ?

P2 : On a un bon système de suivi.

Interviewer : D'accord.

P2 : (Audio inaudible).

Interviewer : D'accord. Le GLPI ce n'est pas le système de réservation de voyage ?

P2 : Non.

Interviewer : Non ? C'est un système à part ?

P2 : C'est un système de laboratoire pour déposer la demande et suivi de demande.

Interviewer : D'accord. Donc, vous travaillez avec deux systèmes. Un système de réservation de voyage et après le GLPI.

P2 : C'est ça.

Interviewer : D'accord. Et même ensuite vous utilisez l'Excel et le GLPI aussi ?

P2 : Oui !

Interviewer : D'accord. Donc.

P2 : Le GLPI c'est justement si la demande est, par exemple, dans le même jour. En cas d'urgence... (audio inaudible) ... mes collègues peuvent intervenir pour traiter les urgences. Un petit problème avec les réservations, on ne peut pas donner la main... (audio inaudible).

Interviewer : D'accord. Et ils voient la demande pour le système de réservation ?

P2 : Ils voient par le GLPI. Heureusement que dans le GLPI provoque une demande, mais sinon, s'ils envoient à moi, non. Si on reçoit par mail ils ne vont pas le voir. Donc, ce qui est intéressant c'est qu'on puisse donner une habilitation à une personne pour traiter une demande de réservation en fait quand on n'est pas là.

Interviewer : D'accord. Et Il n'y a aucune intégration dans ces deux systèmes-là de réservation et le GLPI ?

P2 : Non.

Interviewer : Non ? Toutes les infos doivent être... (audio inaudible).

P2 : Oui.

Interviewer : Bon. Pouvez-vous fournir quelques exemples de demande de réservation que vous recevez ?
*

P2 : Demande de réservation ?

Interviewer : Demande de réservation de voyage, comment cette demande arrive ? En quel format ? Et.

P2 : Dans un format, c'est un portail.

Interviewer : Quel type d'information que vous êtes...

P2 : La ville, les horaires, ce qui est important de voir, la date de confirmation, parce qu'on a un « *delay* » pour confirmer ce voyage. Sinon... (audio inaudible) ... évidemment c'est la compagnie qui met de date... (audio inaudible) ... parce qu'on ne peut pas garder ce vol, parce que, du coup, plus réserver. Quoi qu'on ne puisse pas le garder pendant deux mois, donc on a de date, c'est ça.

Interviewer : C'est un numéro automatique que le système envoie ?

P2 : Oui.

Interviewer : Le système de réservation ? Pas le GLPI ?

P2 : Oui !

Interviewer : Donc. La ville, les dates.

P2 : Les villes, les dates, les horaires, le « *delay* », les prix, les frais d'agence, c'est ça.

Interviewer : Après pouvez-vous m'envoyer un exemple de ce type de demande ? Bien sûr, en élevant le nom, les infos...

P2 : Bien sûr.

Interviewer : Bon sur le traitement des demandes.

Section B : Traitement des demandes de réservation.

Interviewer : Quelle est la procédure-type de traiter une demande de réservation ? Quel est le processus... tout d'abord on fait ça et après ça... *

P2 : Donc. Alors, la première chose à faire c'est déposer sur le GLPI la demande d'ordre de mission... (audio inaudible) ... ensuite quand vous rentrer sur le site de réservation, un document est déjà créé, parce qu'il prend des infos à la base de l'université, CNRS, voilà. Donc, souvent il demande de mettre en jour, parce qu'il manque des infos (date de naissance, etc.) ... audio inaudible... et ensuite la recherche... (audio inaudible) ... la destination que vous souhaitez, une proposition de tarif sans bagage, la classe : première classe, deuxième classe et etc. Et là vous choisissez un tarif que vous voulez, si c'est remboursable ou pas et ensuite, à la fin de votre sélection ou de votre validateur, là vous avez leur destinataire... (audio inaudible) ... et là vous recevez par mail la demande d'autorisation.

Interviewer : D'accord. Donc, vous commencer ce processus-là, après avoir reçu le mail d'utilisateur, c'est ça ?

P2 : Non, là ce le cas où le chercheur fait sa réservation de date sur la demande d'accord.

Interviewer : D'accord ! Dans ce cas-là, il n'y a pas de soucis. Si tout va bien il fait, le chercheur fait la demande, vous recevez le mail avec la demande et donc, c'est après ça que vous commence à saisir les infos dans le GLPI ?

P2 : Non. Le GLPI c'est la base. C'est d'abord le GLPI, c'est avec l'ordre de mission.

Interviewer : Donc, le chercheur fait la demande sur le GLPI directement ?

P2 : Parce qu'il fait la demande d'ordre de mission.

Interviewer : D'accord.

P2 : Avec le justificatif de déplacement, parce qu'on ne peut pas réserver le billet si ce n'est pas justifié. Donc, voilà, en parallèle il va faire sa demande de réservation. Donc, nous, on va la recevoir et si on a toutes les choses sur le GLPI on va faire le bon de commande et... (audio inaudible).

Interviewer : D'accord. Donc, et à quel moment le système de réservation le « *Travel* », je ne sais pas quoi, il entre en scène en fait ? Dans quel moment on utilise ce système-là ?

P2 : Dans la réservation du transport.

Interviewer : Mais avant ou après le processus GLPI ?

P2 : Alors, nous disons au même temps ou après, mais pas avant.

Interviewer : D'accord. Donc, on commence toujours par la demande dans le GLPI et en parallèle ou après, on fait la demande sur le système de voyage, après que le voyage soit déjà approuvé par le responsable, etc.

P2 : Voilà. C'est ça !

Interviewer : D'accord.

P2 : C'est que nous justement...la demande d'ordre de mission et assigné, justifié avec un programme... (audio inaudible) ... là c'est bon pour réserver.

Interviewer : D'accord.

P2 : Audio inaudible.

Interviewer : D'accord. Quelle suggestion pour améliorer ce processus de traitement de la demande ?

P2 : Voilà ! Non, c'est très bien... (audio inaudible).

Interviewer : D'accord ! Est-ce que dans les demandes des voyageurs, que vous traitez, il y a des informations qu'on pourrait identifier comme des besoins et/ou des exigences pour améliorer un système de réservation de voyage ? •

On suppose qu'on va commencer à construire un système de réservation de voyage. Est-ce que dans cette demande vous identifiez des exigences utilisateur, des besoins utilisateur. Comment on peut utiliser comme source pour ce type de système.

P2 : Pas encore.

Interviewer : Non ?

P2 : Non.

Interviewer : Vous pensez que dedans il n'y a pas d'informations utiles pour aider la construction d'un système dans ce type-là ?

P2 : Malheureusement je ne vois pas.

Interviewer : Non ?

P2 : Non.

Interviewer : Même sur le problème que vous recevez, par exemple, les chercheurs, ils n'arrivent pas à chercher les vols qu'ils veulent. Donc, même dans ce cas-là, vous ne pensez pas qu'il y a des choses qu'on doit porter pour améliorer ce type de système ?

P2 : Je pense qu'après c'est voulu. Parce que voulus ne mettent pas tous les vols en ligne. Les vols compliqués, il y a des frais d'agence, peut-être, du coup... je ne sais pas... il y a beaucoup de destination... Oui... peut-être on peut améliorer ça en cas de vols compliqués.

Interviewer : Donc, vous pensez qui ça peut être une chose qu'on peut identifier comme besoin utilisateur, que le système doive prendre en compte ?

P2 : Oui !

Interviewer : C'est un problème.

P2 : On fait la demande par mail... (audio inaudible).

Interviewer : À cause de ce « *delay* » il n'y a pas de problème ?

P2 : On dit que ce « *delay* » ... En ligne, on valide tout ensuite, il n'y a pas de « *delay* ».

Interviewer : D'accord. Quelles sont vos besoins/exigences pour le système de réservation de voyage que vous utilisez actuellement ? •

Quelque chose que vous voudrez avoir dans ce type de système, ou que vous considérez essentielle ?

P2 : Non, là c'est assez complet.

Interviewer : Quel type de fonctionnalité dans le système vous considérez plus important ?

P2 : Fonctionnalité ?

Interviewer : Oui ! La partie, par exemple, d'ordonner le vol par prix, c'est important ou non, par exemple ?

P2 : C'est important !

Interviewer : Il n'y a pas de fonctionnalité que vous considérez...

P2 : Tirer par horaires... (audio inaudible) après les informations des passeports.

Interviewer : D'accord ! Avec quelle fréquence vous devez demander l'aide des autres membres de l'équipe pour résoudre les demandes des voyageurs ? *

P2 : Les autres membres de l'équipe ?

Interviewer : Oui, si vous avez besoin, bien sûr.

P2 : Oui ! Mois, tous les jours. Nous sommes dans un « *open space* », donc on pose les questions naturellement... (audio inaudible) ... la fréquence...

Interviewer : Tous les jours, peut-être ?

P2 : Voilà, tous les jours.

Interviewer : Donc, c'est assez fréquent. Quelque suggestion pour faire ça mieux ?

P2 : Non !

Interviewer : Non ?! Avec quelle fréquence vous devez demander aux voyageurs de clarifier les informations concernant leur demande ? *

P2 : On est toujours obligé de demander, sur tout par rapport au service au GLPI, parce que, après la réservation dans le même service est claire, pas de problème. Tandis que ce nous qui faisons... (audio inaudible). Quand ce le chercheur qui fait sa réservation, c'est sûr, c'est clair. Après ce juste par rapport au document rempli, l'émission d'ordre de mission, etc.

Interviewer : D'accord ! Savez-vous me dire avec quelle fréquence ? Tous les jours, tout le temps ?

P2 : Deux à trois fois par jour.

Interviewer : D'accord. Quelque suggestion pour faire ça mieux ? Pour améliorer la clarification de problème avec les voyageurs ? Quelque fonctionnalité ?

P2 : Laissez de se communiquer avec la... (audio inaudible). En ligne, sur l'internet a une procédure d'utilisation. On fait passer des messages, voilà, sauf, je pense que beaucoup de personnes ne réalisent pas et voilà.

Interviewer : D'accord !

P2 : On a fait le service général, quand ils ont mis quelque... (audio inaudible). En place, il avait une dizaine de personne... (audio inaudible) ... donc, voilà, on ne sait pas trop comme on fait pour savoir quel vous intéresse et voilà.

Interviewer : Selon votre propre expérience, quelles seraient les fonctionnalités qui vous seraient utiles et qui devraient être rajoutées au logiciel de réservation ? •

P2 : Pour la réservation, je ne dirais rien.

Interviewer : Non ? D'accord ! Aucune. Selon votre propre expérience, quelles seraient les fonctionnalités qui seraient utiles pour les voyageurs ? •

P2 : Que seraient quoi ?

Interviewer : Utile pour les voyageurs. Dans le système de réservation de voyage, il y a quelque fonctionnalité que vous pensez qui pourrait aider les voyageurs à faire le processus de réservation de manière plus facile ?

P2 : C'est très simple. Je pense que c'est comme une réservation sur l'internet. C'est pareil.

Interviewer : C'est pareil avec tous les autres qu'on a déjà sur l'internet ?

P2 : C'est très bon. On registre notre destin, les horaires, vous sélectionnez et juste à la fin, mettre le vol et c'est très simple. Souvent, on se retrouve sur les vols « *EasyJet* », justement parce que... (audio inaudible) Ils ne savent pas, nous ne pouvons pas vous renseigner... (audio inaudible). Oui, c'est normal, parce que on ne peut pas le faire.

Interviewer : D'accord.

P2 : (Audio inaudible).

Interviewer : Donc, le problème c'est plutôt avec les « *low-cost* »

P2 : Oui !

Interviewer : Pourriez-vous lister 3 fonctionnalités que vous aimerez garder pour ce type de système ? Le système de voyage, pas le GLPI. •

P2 : Oui ! Tirer par horaire et par le prix, c'est super ! L'effet que le profil est complet... (audio inaudible). Le profil est déjà et complet et voilà. Quoi d'autre ? Quoi d'autre ? Je ne vois pas plus.

Interviewer : Non ?

P2 : Non.

Interviewer : D'accord ! Pourriez-vous lister trois fonctionnalités que vous aimerez changer pour ce type de système ? Faire de manière différente. Rajouter la question de multiples-destinations ce ne pas une bonne fonctionnalité, que vous avez dire •

P2 : Oui, oui ! Pour les multiples-destinations, ça c'est sûr que ce n'est pas évident. Après ça, c'est tout simple.

Interviewer : D'accord ! Très bien.

Section C : Rédaction des Récits Utilisateurs.

Interviewer : Bon ! Dans cette partie concernant la rédaction, que nous, on s'appelle le récit utilisateur. C'est un « *template* », c'est un modèle pour écrire les histoires, les récits, quand, l'utilisateur fait une action sur le système. D'accord ?

Donc, le modèle c'est plutôt comme ça. On a un titre, d'accord ? Que décrit l'histoire et on a un préambule avec un rôle qui fait cette fonction-là. Qu'est-ce qu'il veut comme fonctionnalité, afin de voir quelque but, quelque bénéfice. Voilà ! Et Donc, on a plusieurs scénarios pour décrire plusieurs situations qu'on peut utiliser avec le système. Donc, on a toujours une clause « **En tant donnée** », qui va nous donner un contexte d'application de ce scénario. On peut avoir plusieurs contextes, donc on rajoute la clause « **Quand** » un évènement arrive et une conséquence « **Alors** », cette chose ou plusieurs choses arrivent, donc un exemple :

En tant donné que je vais à la page “Recherche des vols”

Quand je choisis : “aller simple”

Et je tape “Paris” et choisis “Paris, Charles de Gaulle (CDG)” dans le champ “Départ de”

Et je tape “Toulouse” et choisis “Toulouse, Blagnac (TLS)” dans le champ “Arrivée à”

Et je choisis “2” dans le champ “Nombre total de passagers”

Et je choisis “15/12/2017” dans le champ “Date de départ”

Et je clique sur “Recherche”

Alors le système va afficher la liste des vols disponibles.

Donc, la question que je pose : est-ce que vous pourriez spécifier un exemple des demandes de réservation en utilisant ce format-là description ? Bien sûr, différent de celui-là ?

P2 : Elle lit.

Interviewer : Ça vous semblez comment ce format-là ?

P2 : Oui, c'est ça ! C'est un peu ce contexte en fait.

Interviewer : Donc, c'est quoi ? C'est un modèle que vous considérez qu'on peut faire, qu'on peut écrire l'activité utilisateur comme ça ?

P2 : (Audio inaudible).

Interviewer : Par exemple, si on veut chercher un vol multiples-destinations quoi vous me donnez comme exemple ? Qui est le problème !

P2 : Oui !

Interviewer : Vous arrivez le décrire comme ça ?

P2 : Oui !

Interviewer : Oui ? Bon, si vous pouvez le faire un exemple comme ça.

P2 : Oui !

Interviewer : Oui ! En vers qu'en suivant ce modèle-là. Vous devez maintenir juste les clauses que sont là, d'accord ? Ici, bien sûr, vous pouvez tout changer.

P2 : (Elle écrit).

Interviewer : On doit, par exemple, arriver hors Paris. Sortir de Frankfurt...

P2 : Parce que de toute façon, par exemple, les multiples-destinations...

Interviewer : Par exemple.

P2 : (Audio inaudible).

Interviewer : Il n'y a pas une option multiples-destinations ?

P2 : (Audio inaudible).

(Elle écrit en lisant quelque chose en voix bas).

Interviewer : C'est comme plusieurs allers-simples ?

P2 : On fait des allers-retours aussi... (audio inaudible).

Interviewer : Mais dans cette même interface-là ?

P2 : Oui... (Elle écrit encore).

Interviewer : D'accord ! Donc, c'est encore plus compliqué, si par exemple, il fait Toulouse, Rio et il ne part pas de Rio, il part de São Paulo à Porto Alegre, par exemple ?

P2 : Oui !

Interviewer : Il faut faire d'aller-simple. Il ne peut pas faire ça comme multiples-destinations ?

P2 : Non !

Interviewer : Donc, Il faut partir dans la même ville que vous êtes arrivée ?

P2 : Comme ça.

Interviewer : Oui ?

P2 : Après c'est pareil.

Interviewer : D'accord ! Normalement les tarifs sont plus chers pour les allers-simples que pour les multiples-destinations. Donc, ça pose beaucoup de problèmes.

P2 : Oui !

Interviewer : Bon, c'est bon ? Il faut juste une conclusion, je pense.

P2 : (Elle écrit après elle a lu en voix bas).

Interviewer : D'accord ! Bon, la dernière partie, pensez-vous pouvoir rédiger une liste de demande/problème, surtout de problème que vous recevrez au cours de la semaine que va venir ? Que les utilisateurs rencontrent sur la recherche du vol etc. ?

P2 : Oui.

Interviewer : Une liste simple, avec bon, ça c'est un problème, ça c'est d'autres problèmes qu'ils ont racontés.

P2 : Oui !

Interviewer : C'est possible ?

P2 : Oui, bien sûr.

Interviewer : Vous avez une idée de combien de problème vous avez normalement pour la semaine ? Que vous recevoir ?

P2 : Deux par semaine.

Interviewer : Deux ? Donc, ce n'est pas beaucoup.

P2 : Ce n'est pas beaucoup, mais ça dépend de période. Souvent les plus compliqués c'est quand on a des invités que viennent des pays qui sont très loin.

Interviewer : D'accord ! Donc, si Oui, c'est le cas. Pensez-vous que vous pouvez formater ces demandes/problems, en suivant ce modèle-là ?

P2 : Oui !

Interviewer : Oui ? Par exemple, on n'arrive pas mettre le parcours de voyage donc, c'est un problème, donc, vous-pouvez arriver à formater la demande à c'est format-là ? Vous-croyez que c'est possible ?

P2 : Oui ! Vous voudrez que je rédige

Interviewer : Une liste de problèmes que vous rencontrez et après formater dans c'est format-là.

P2 : Oui !

Interviewer : Oui ? C'est possible ? Donc, dans ce cas-là, je veux vous envoyer par courriel électronique, la semaine prochaine peut-être.

P2 : C'est un temps court.

Interviewer : Oui ! Désolé !

P2 : Pas de souci.

Interviewer : Et après ça, je vais vous envoyer un bref questionnaire pour évaluer ce type de « *template* ». D'accord ?

P2 : Oui

Interviewer : Donc, ton adresse mail ?

P2 : Oui !

Interviewer : Donc, on fait comme ça. Voilà ! Merci beaucoup.

3. TRANSCRIPTION : Participant 3 (P3)

Partie I : Questionnaire démographique et de contexte.

Interviewer : Bon, voilà ! Donc, la première partie, c'est une partie démographique, on veut savoir votre sexe, c'est évident.

P3 : C'est évident - Féminin.

Interviewer : Votre âge ? S'il vous plaît.

P3 : 52 ans

Interviewer : Votre niveau d'étude ?

P3 : Bac

Interviewer : Depuis combien de temps, vous êtes au service de mission de l'IRIT ?

P3 : Bientôt quatre ans.

Interviewer : Avez-vous déjà eu des expériences dans des services similaires auparavant ?

P3 : Non, pas pour de réservation. Non, non.

Interviewer : Bon, pourriez-vous nous donner un aperçu de ce travail, en fournissant une brève description de vos tâches ?

P3 : Ben ! On reçoit la demande par ticket en fonction de la demande, de la date, de l'heure de la destination nous, on va faire une demande devis qu'on reçoit un peu du temps après et qu'on confirme par bon de commande.

Interviewer : D'accord et vous êtes en charge de tous les tâches pendant le processus. Il n'y a pas de tâche spécialisée pour chacun ?

P3 : Non, non. On fait tous la même chose et on traite chacun un certain nombre d'équipes, voilà.

Interviewer : Et comment on fait la distribution de demande ?

P3 : De demande ?

Interviewer : Oui. Qui prend quoi ?

P3 : Ah ! C'est par équipe, par exemple vous, vous faites partie d'une équipe, la gestionnaire de cette équipe va récupérer la demande et elle va traiter la demande de la mission, du billet d'avion, de train jusqu'au retour de mission. C'est tout !

Interviewer : D'accord ! Mais ma demande qui prend charge ? Vous, P1, Lorraine ? Il y a un sort de tâche spécialisée qui quelqu'un fait de l'équipe ?

P3 : Non ça dépend des comptes, sinon, chacune est responsable d'un nombre de compte.

Interviewer : Ah ! D'accord ! Vous êtes responsable par un sort de compte ?

P3 : Oui.

Partie II : Processus de traitement de demandes

Nous sommes intéressés par les préférences et les difficultés que les voyageurs de l'IRIT ont rencontrées et vous ont signalés lorsqu'ils essaient de réserver leurs voyages d'affaires. Nous sommes également intéressés par votre opinion sur les demandes reçues.

Label : * faits, • interprétation

Section A : Réception de demandes de réservation

Interviewer : D'accord ! Bon, sur le processus de traitement de demandes - À la réception de demandes de réservation : Comment les demandes de réservation des voyageurs arrivent-elles à vous et avec quelle fréquence ? Avez-vous des suggestions pour faire mieux ? *

Interviewer : Bon, elle arrive par ticket !

P3 : Par ticket, voilà !

Interviewer : C'est toujours par ticket ?

P3 : Ou par mail, mais le plus souvent c'est par ticket, parce qu'on essaie de le mettre en place le plus possible, pour avoir une visibilité plus générale, justement entre les gestionnaires. Parce qu'on a l'absence du gestionnaire justement, du ou la gestionnaire qui pourrait être malade ou absente, pour quelque motif que c'est soin, sa collègue ou son collègue peut reprendre le ticket pendant ces temps-là, comme ça, s'il y a une urgence, il ne passera pas l'attrape, on va le gérer, on ne va pas attendre que la gestionnaire, qui était absente, va revenir pour traiter ces tickets.

Interviewer : D'accord ! Ils arrivent dans quelle fréquence ? En Moyenne.

P3 : De ticket, on a tous les jours, de demande !

Interviewer : Un peu prêt ? Combien ? Vous savez dire ? En Moyenne ?

P3 : En moyenne ? Je dirai entre 5 et 10 par jour.

Interviewer : D'accord ! Avez-vous des suggestions dans ce processus ? De comment la demande arrive ?

P3 : Non, ça fonctionne très bien.

Interviewer : D'accord ! Bon, combien de demandes de réservation de voyage avez-vous reçues la semaine dernière ? *

P3 : On va dire, de réservation de voyage. Je dirai 10.

Interviewer : Donc, ça c'est une semaine typique. Parce que bon...

P3 : Ça fluctue, parce que, quand tu as de soutenance de thèse, ben ! On a plus à ce moment-là, après on a en moins, après on peut avoir une semaine où il y a deux soutenances, donc, ça fait beaucoup plus de demande, donc ça fluctue vraiment.

Interviewer : D'accord !

P3 : Mais, on a toujours au minimum, je pense, 5 par semaine au minimum.

Interviewer : Avez-vous des suggestions dans cette partie, sur la demande ? *

P3 : Sur la demande des chercheurs ?

Interviewer : Oui ! Si elles viennent tous ensemble...si...bon, je ne sais pas !

P3 : Elle ne peut pas venir tous ensemble, parce qu'ils n'ont pas les démarches au même moment. Parce que chaque jour a sa propre organisation, donc, mais s'il peut anticiper le maximum, ça serait meilleur, parce que chacun ne pas "c'est le monde", donc, il faut deviner et savoir que nous on n'attend pas cette demande pour travailler. On fait les demandes, on fait la mesure qu'elles arrivent. Souvent pour une demande on perdre beaucoup de temps, parce que dans la demande il n'y a pas que le billet d'avion il y a aussi les gens qui ne sont pas créés, donc, par exemple, donc ça fait la demande de document mal rempli, c'est que retarder la demande de temps et il ne faut pas faire. En général, quand l'agent fait la demande vendredi, mais on ne peut pas forcément le temps de le valider le vendredi même. Donc, si vous, c'est un exemple. Vous faites une demande de devis le vendredi après-midi à 16 h, nous, on part 16h30, maximum 17h30. Si on traite de demande d'urgence avant de recevoir votre ticket, le devis que vous allez demander, si vous regardez bien, sur votre devis, ça être validé avant le même jour, minuit par exemple, on ne peut pas valider. C'est qui fait que le lundi, vous allez être obligé de faire une nouvelle demande de devis, donc voilà, il faut faire très attention à la date de validité sur la demande de devis.

Interviewer : D'accord ! Pensez-vous qu'il manque quelque chose dans la description des demandes de réservation que vous recevez ? •

P3 : En ce qui me concerne, non.

Interviewer : C'est complet ? Ne manque aucun donné ?

P3 : Non pas, il ne manque rien ! En règle générale, non !

Interviewer : D'accord ! Avez-vous de suggestion à ce sujet-là ? •

P3 : Oui, j'ai juste une suggestion à faire. Si l'agent nous envoie des horaires précis et alors, que dans sa tête : « c'est bon, je mets ces horaires-là, mais ça peut changer ». Il faudrait qu'ils le président dans sa demande, parce que nous, on va se précipiter pour faire la réservation et si après l'agent va dire : « finalement non, finalement, ça me range plus une heure après ou une heure plus tôt ». Nous, on va retravailler dessus, ça va faire perdre de temps, par une heure du temps. Alors que s'il y a un moins de doute, qu'ils précisent au départ, ça, nous éviterons de nous précipiter et de perdre de l'argent aussi parce que souvent pour une modification, il y a une pénalité, voilà.

Interviewer : La plupart des demandes arrivent avec un horaire fixe. C'est ça ?

P3 : Oui, oui.

Interviewer : Donc, si on peut avoir des horaires flexibles, ça sera mieux ?

P3 : Pour la personne qui a un doute, ils vont mieux avoir des horaires flexibles, oui.

Interviewer : D'accord ! Il y a des champs comme ça, pour faire cette option d'horaire flexible dans l'ouverture de ticket ? Ou il faut l'écrit ?

P3 : Non, il faut l'écrit.

Interviewer : D'accord.

P3 : Quand c'est nous qui faisons la demande de devis. Après quand c'est vous qui la faites, là, si vous engagement. Vous, propre, après une fois qu'on a le devis que vous, vous avez demandé on a obligé de suivre vous horaire, mais dans le ticket, si vous avez un doute, il faut le préciser, voilà.

Interviewer : D'accord.

P3 : Ça sera meilleur, ça, nous faisons gagner du temps et à vous de l'argent.

Interviewer : D'accord ! Est-ce que vous devez prendre des notes (ex. post-it, email etc.) sur les demandes de réservations ? *

P3 : Non.

Interviewer : Non ?

P3 : Non

Interviewer : Bon, si oui, combien de notes en moyenne ? Donc, ça ne fait pas de sens. *

P3 : On ne peut pas de note, parce qu'on a deux écrans, donc, si on a besoin d'information, on peut travailler, on peut le réserver sur un écran puis on peut regarder les horaires sur la demande sur l'autre écran. Donc comme ça, on n'a pas besoin de noter. Sur un doute, on regarde la demande.

Interviewer : D'accord ! Mais, par exemple, même si vous prenez une demande par téléphone ou d'information, il n'y a pas de note que vous prenez ? En dehors du système ?

P3 : Voilà, quand on a tout le suivi de toutes les demandes, on peut remonter jusqu'au départ. On ne prend pas des notes par téléphone, c'est trop fragile. S'il y a au moins d'erreur, après, on n'a aucune trace.

Interviewer : Même s'il y a une demande formalisée par ticket et il y a un doute, par exemple ?

P3 : Non, toujours par écrit.

Interviewer : Donc, toujours par écrit ?

P3 : Toujours par écrit, parce que dans le ticket, on peut faire un suivi.

Interviewer : D'accord !

Interviewer : Quelque suggestion dans ce processus-là ?

P3 : Non, ça fonctionne très bien, je pense. Je n'ai jamais rencontré, au niveau des demandes, des agents. À part le fait qu'il faut qu'il soit sûr de leurs horaires, quand ils demandent un devis. Bon après, on sait qui peut avoir les contremorts de dernière minute, ça arrive à tout le monde, mais voilà. Le plus possible évité de nous refaire travailler plusieurs fois sur une même demande.

Interviewer : D'accord ! Vous prenez les annotations, tout ça, sur le suivi de ticket, c'est ça ?

P3 : Oui.

Interviewer : D'accord ! Et vous avez une idée du combien de suivi en demande elle prend normalement ?

P3 : De suivi ? Ça varie aussi. On ne peut pas, je ne sais pas si on peut le chiffrer, parce qu'il peut avoir un ou deux problèmes et puis le suivant, il va y avoir un échange de mail demande, quatre textes pour une seule demande, par exemple. Donc je ne sais pas si on peut vraiment le chiffrer. Voilà, ça varie vraiment en fonction de la personne, de la demande et voilà, je pense que c'est ça.

Interviewer : D'accord ! Si vous prenez des notes, comment vous les conservée et sur quel format ? Donc, vous le conservées sur le suivi de demande, c'est ça ? *

P3 : Oui, et on fait souvent une copie papier dans le dossier.

Interviewer : Physiquement ? Vous imprimez ?

P3 : Oui

Interviewer : Comment améliorer l'enregistrement de ces notes ? Il y a quelque chose ?

P3 : Non, parce que... Il n'y a aucune, le suivi. Je ne sais pas si vous voyez. Vous avez déjà regardé ?

Interviewer : Oui, Oui. GLPI, c'est ça ?

P3 : GLPI, voilà ! Non, et vous cliquez sur le traitement de texte, je crois, et vous avez tous les suivis que s'affiche, ainsi que les documents.

Interviewer : Je ne connais pas l'interface de votre côté, mais bon.

P3 : Mais c'est bien ! On a tous les suivis.

Interviewer : D'accord ! Très bien ! Pensez-vous que l'enregistrement de ces notes est important ? •

P3 : Oui, c'est très important !

Interviewer : Bon, c'est important, mais il n'y a pas de suggestion d'amélioration dans cet enregistrement ?

P3 : Non, pas ce qui me concerne.

Interviewer : D'accord.

Interviewer : Pouvez-vous fournir quelques exemples de demande de réservation que vous recevez ? Par exemple, c'est un voyage qui apporte plusieurs moyens des transports, il y a une partie en avion, une partie en train et c'est tout dans le même billet. *

P3 : Il y a plusieurs destinations, il y a de billet d'avion avec de changement d'aéroport.

Interviewer : C'est surtout de billet d'avion et de train ?

P3 : Oui.

Interviewer : D'accord. Par exemple, il y a de réservation de la voiture ?

P3 : Oui, il y a de location de la voiture aussi.

Interviewer : D'accord. Au niveau du système, ça, c'est traiter de la même façon ?

P3 : Oui, sur l'interface, on peut réserver.

Interviewer : D'accord. Quelque suggestion à ce sujet-là ? D'amélioration ?

P3 : Oui, d'amélioration, oui. J'ai eu un problème la semaine dernière, parce que j'ai eu un vol pour aller sur une compagnie et le retour sur une autre, et le retour, c'est sur un vol « *low-cost* ». J'ai dû faire une demande indirecte au marché, je ne suis pas passé par le site, parce que, je n'ai pas trouvé, il ne voulait pas prendre les deux compagnies en même temps j'étais obligé de prendre le vol séparément, or séparément c'est très cher, il ne voulait pas faire aller et retour. Donc, j'étais obligé de faire une demande par mail au marché, qu'eux m'envoyer un devis, alors que, c'est plus rapide, voilà.

Interviewer : D'accord, quand vous dites au marché, c'est l'agent de voyage ?

P3 : Oui, l'agent de voyage.

Interviewer : D'accord. Il fait ça et il vous retourne un billet aller-retour avec deux compagnies différentes, c'est ça ?

P3 : Oui, c'est le devis qu'il nous envoyait avec la compagnie qu'on souhaitait. Par exemple, vous, vous allez regarder sur l'internet, vous allez voir, le départ, je peux prendre Air France et le retour, je peux revenir avec « *EasyJet* ». Vous voyez que les horaires vous conviennent, donc, vous m'envoyez une demande, moi, je vais sur le site, je regarde les horaires et les compagnies. Je ne vois pas la compagnie « *EasyJet* » sur leur site. Bon, on dit ! Vous n'avez pas inventé quand même.

Interviewer : Mais, ça arrive toujours ? Les « *low-costs* » ne sont pas concernés dans le système ?

P3 : Non, on a des « *low-costs* ».

Interviewer : Mais pourquoi vous ne voyiez pas ?

P3 : Parce que, quand il y a de vol aller-retour, on ne prend pas forcément le « *low-cost* », donc ça va perturber, je pense.

Interviewer : D'accord, donc vous croyiez que c'est un problème du système plutôt ?

P3 : Je pense que... Je ne sais pas. Non, en tout cas quand leur demande, quand moi, je vais faire la demande par mail à l'agence, voilà, il m'envoie exactement les horaires et les compagnies que vous, vous m'avez demandé, donc, c'est possible pourquoi on n'arrive pas à voir sur le site. Donc ça, vous faites perdre du temps.

Interviewer : D'accord. Donc, la suggestion c'est de pouvoir faire ça sur le site ?

P3 : De pouvoir faire ça sur le site.

Interviewer : D'accord.

Section B : Traitement des demandes de réservation.

Interviewer : Bon, concernant le traitement de demandes. Quelle est la procédure-type pour traiter une demande de réservation ? *

P3 : Alors, la procédure type.

Interviewer : Comment ça commence, après qu'est-ce qu'arrive ?

P3 : Uniquement de la réservation ou de la globalité ?

Interviewer : De voyage, de voyage, surtout.

P3 : Bon, l'agent nous met un ticket GLPI. C'est lui qui n'y a pas l'accès. Parce qu'il y a deux possibilités. Il y a de gens doctorant qui il n'y a pas accès à l'intranet qui ne pouvez pas faire leur demande de devis, donc il passe directement par nous.

Interviewer : Par mail ?

P3 : Il faut un ticket.

Interviewer : Mais c'est un ticket différent des autres qui ont d'accès ?

P3 : Non, c'est toujours GLPI. Sauf que vous, si vous êtes, si vous avez-vous identifiant intranet UPS, vous pourrez faire directement votre demande de devis et nous mettre en tant que valider.

Interviewer : D'accord.

P3 : Donc nous, on va recevoir la demande, d'accord ?

Interviewer : D'accord. Donc, ça c'est le premier part.

P3 : Voilà, après il y a les gens qui n'ont pas leur identifiant, qui vont être obligés de passer par nous. Donc, ils vont faire un ticket également, ils vont nous transmettre la date, la destination et les horaires par mail. Nous, on va les enregistrer en tant qu'inviter, parce que, ils ne sont pas créés. Et on va faire la demande. On va réserver pour eux et on va demander un devis.

Interviewer : D'accord. Et la demande de devis, comment ont fait cette demande ? Vous allez au site ? C'est ça ?

P3 : Voilà.

Interviewer : Ou vous appelez l'agence ?

P3 : Non, non. Je vais sur le site.

Interviewer : C'est toujours sur le site ?

P3 : Oui, toujours en priorité c'est sur le site.

Interviewer : D'accord. C'est le « *Travel Planet* ».

P3 : Oui, « *Travel Planet* ».

Interviewer : D'accord. Donc après le devis ?

P3 : Après le devis, donc, on fait une demande de devis qu'on va recevoir par mail et qu'on va confirmer par un bon de commande. Bon, nous, on fait le bon de commande et sur le site on va repartir pour finaliser le voyage, finaliser la demande, donc on va...

Interviewer : Finaliser ça veut dire quoi, exactement ?

P3 : Finaliser c'est donner notre accord par numéro de bon de commande. Voilà. Au départ, on demande l'accord par devis. Une fois, qu'on a le devis, on va finaliser la commande en précisant le numéro de bon de commande qui a attaché à cette demande, comme ça, eux n'a pas besoin de recevoir le bon de commande, ils ont juste le numéro pour attacher la facture, le billet à ce numéro.

Interviewer : D'accord ! Et après ça c'est fini ?

P3 : Après on reçoit l'itinéraire, qu'on soit en le reçois uniquement nous et n'est pas l'agent ou alors, l'agent reçoit aussi une copie. Mais, nous, par défaut, l'envoie quand même dès qu'on reçoit l'itinéraire on le renvoie à l'agent pour être sûr qu'il puisse enregistrer en ligne.

Interviewer : Que c'est déjà le billet ?

P3 : Ce n'est jamais le billet en général, c'est pour qu'il s'enregistre en ligne et qu'il imprime son billet.

Interviewer : D'accord.

P3 : On ne reçoit pas immédiatement. Ça, peut-être un déstabilisant, si on peut avoir dans les 48 h, suivant la demande, ça sera bien.

Interviewer : Mais aujourd'hui c'est combien du temps ?

P3 : On le reçoit trois, quatre jours après. Donc, nous, on passe à l'autre chose et puis un jour on reçoit l'itinéraire puis on est obligé de perdre du temps. Ah ! Oui, c'est vrai, on est obligé de ressortir le dossier pour savoir où est le dossier, s'il n'y a pas d'erreur. Voilà !

Interviewer : D'accord !

P3 : Si ça pouvait être plus rapide, ça serait mieux.

Interviewer : Donc la suggestion dans ce cas-là, c'est de pouvoir, par exemple, confirmer la demande directement sur le site ? Ou non ? C'est même de la recevoir dans un « *delay* » plus court ?

P3 : Non, c'est plutôt de la recevoir dans un « *delay* » plus court. Donc, c'est encore frais dans notre esprit.

Interviewer : Est-ce que dans les demandes des voyageurs que vous traitez, il y a des informations qu'on pourrait identifier comme des besoins et/ou des exigences pour améliorer un système de réservation de voyage ? •

P3 : Des besoins, des exigences... moi, je n'ai pas eu. Ou peut-être ?

Interviewer : Par exemple vous voyez quelque sort de problème qu'arrive souvent pour le chercheur, quand ils vont chercher du voyage sur le système. Ça pourrait devenir un besoin utilisateur pour améliorer un système futur de réservation de voyage ? Ou même le système qui existe déjà ?

P3 : Je ne vois pas trop qu'est-ce que pourrait... Bon, moi, j'ai une, que je n'ai pas utilisée. C'est le voyage groupé, avec plusieurs personnes. Ça je n'ai encore fait depuis que... c'était bien en place, donc, je ne peux pas vous en parler, je ne peux pas dire si c'est efficace ou pas.

Interviewer : C'est quoi un voyage groupe exactement.

P3 : Par exemple il y a une dizaine de personnes qui prennent le billet d'avion le même jour. Donc, qui vont à la même conférence, voilà. Et vous voudriez aller dans le même vol et la même heure.

Interviewer : Donc, c'est possible de faire une seule demande pour les dix ?

P3 : Ah, non ! Il faut faire la même, mais... Il faut avoir, si on peut avoir... Non, peut-être pas... Peut-être il faut faire une demande... Je ne peux pas vous en parler, parce que je jamais essayais, je ne sais pas en fait si on peut faire... c'est possible. Je pense que c'est possible de faire une seule demande pour un groupe de personnes.

Interviewer : D'accord. Donc, c'est que concerne le besoin, les exigences, il y a quelque chose que vous identifiez ?

P3 : Les grandes personnes. Sont des exigences dans déplacement, donc, ce n'est pas évident, parce que, sont des grandes jambes et les vols ont des lieux trop précis.

Interviewer : Ça, on ne peut pas faire aujourd'hui ?

P3 : Ça dépend. On peut le faire s'il le prend tout, mais ce n'est pas évident.

Interviewer : Mais vous pouvez demander ça par téléphone, par exemple, à l'agent de voyage que va faire... ?

P3 : Par téléphone, non, Il ne renseigne jamais par téléphone. On a toujours besoin de faire un mail. Il ne donne jamais de réponse par téléphone. Mais en fonction... on a un plan d'emplacement dans l'avion et on voit si c'est disponible ou pas disponible. Quand ne plus disponible, ce n'est pas possible.

Interviewer : Même s'il intervient, l'agence ? Même si l'agence fait une intervention pour... ?

P3 : Oui ! Non.

Interviewer : On ne peut pas. D'accord !

Interviewer : Quelles sont vos besoins/exigences pour le système de réservation de voyage que vous utilisez actuellement ? Par exemple, il y a de fonctionnalité que vous voudriez voir, en ce que concerne votre activité ? •

P3 : Alors, laisse-moi réfléchir. Je voudrais que, peut-être, entre le moment où on a fait la demande de devis et le moment où, on fait le bon de commande, souvent, ce à ce rapide. On retourne sur le site et il ne trouve pas notre demande d'autorisation. Donc on est obligé de repartir à zéro ou d'être obligé de taper la référence pour retrouver le voyage, pour retomber dans la liste d'autorisations. Donc, c'est un peu...

Un vrai lien direct entre le moment où on fait le bon de commande et où on donne l'accord. Il que se passe, un petit peu de temps et après on est obligé de tout, de revenir en arrière et souvent il ne trouve pas ça perturbe, donc on est obligé de retaper la référence du vol, du devis, pour revenir sur l'accord

Interviewer : D'accord ! Et ce que concerne la recherche du vol, l'interface pour trouver de vol disponible, tout ça vous plaît aujourd'hui ? Il n'y a pas de besoins que vous identifiez dans cette partie, par exemple ?

P3 : Non, ça va ! C'est à c'est vaste, quand même. On a de la marge. Oui, oui.

Interviewer : D'accord ! Avec quelle fréquence vous devez demander l'aide des autres membres de l'équipe pour résoudre les demandes des voyageurs ? *

P3 : De l'aide de, de...

Interviewer : De l'aide interne. De votre équipe.

P3 : De mes collègues ?

Interviewer : Oui, oui, oui.

P3 : C'est très, très rare.

Interviewer : Rare ?

P3 : Oui, c'est rare. Très rare.

Interviewer : Il y a de suggestion dans ce sujet-là ? De demander l'aide quand il y a de besoin évidemment ?

P3 : En règle générale quand on a un souci on en parle entre nous et on a la chance d'être en quatre, ce qui fait, que sur le quatre il y a toujours, au moins une personne que rencontrer, peut-être, ce genre de problème et, donc, du coup, le règle...

Interviewer : Ça marche bien ?

P3 : Oui, ça marche bien. En règle générale quand on a un doute, on envoie un mail à l'agence, au marché et puis... on n'a pas la réponse immédiatement, mais ils sont à ce réactifs, quand même.

Interviewer : D'accord. Avec quelle fréquence vous devez demander aux voyageurs de clarifier les informations concernant leur demande ? *

P3 : Ah...Ça c'est toujours par ticket ou préalable, donc.

Interviewer : Mais ça arrive souvent ? Des informations manquent ? Des choses qui ne sont pas très claires ?

P3 : Non, mais souvent que manque en fonction de la destination, ce sont le passeport, les numéros de passeport. Ça c'est important. Parce qu'on sait qu'il faut le passeport pour voyager, au moment de prendre son vol, mais pour certaines destinations s'il n'y a pas le numéro du passeport ils ne vont pas le valider.

Interviewer : D'accord.

P3 : Donc, ça il faut que ce sache.

Interviewer : D'accord ! Mais au niveau de fréquence vous arrivez d'identifier ? À quelle fréquence vous avez de demander une clarification ?

P3 : Non, on a très peu.

Interviewer : Donc, c'est rare aussi ?

P3 : Oui, c'est rare.

Interviewer : D'accord. Des suggestions de ce sujet-là ? Comment vous pouvez faire cette clarification au demandeur ? Comment on peut faire ça meilleur ?

P3 : Bah ! Peut-être qu'il a un petit truc qu'il dit à l'agent, au moment de faire sa réservation, de vérifier son profil, de mettre à jour le profil systématiquement avant chaque demande.

Interviewer : D'accord. Selon votre propre expérience, quelles seraient les fonctionnalités qui vous seraient utiles et qui devraient être rajoutées au logiciel de réservation ?

P3 : Bah ! Comme c'est l'internet, un question un conseil, ça ne sera pas mal. Quelqu'un qui sont en direct en ligne.

Interviewer : D'accord. Un *chat* ?

P3 : Un *chat*, voilà !

Interviewer : Avec le demandeur ?

P3 : Bien sûr. Ou nous, ou nous, si on est bloqué, au moment donné, plutôt que de perdre du temps d'envoyer un mail, ou d'attendre. Poser la question directe en ligne, avec l'écran.

Interviewer : Avec l'agence ?

P3 : Et voilà, avec l'agence.

Interviewer : D'accord. D'autres choses ?

P3 : Non, parce que c'est bien fait.

Interviewer : Vous considérez que c'est complet le système ?

P3 : Pour nous besoins, je pense que oui. Bon il y a des améliorations à apporter, mais...

Interviewer : Par exemple ?

P3 : Je ne sais pas, han ! Il y a toujours des améliorations.

Interviewer : Vous n'avez pas identifié à ce moment ?

P3 : Pas vraiment. Pas quelque chose que soule les yeux.

Interviewer : Selon votre propre expérience, quelles seraient les fonctionnalités qui seraient utiles pour les voyageurs ? •

P3 : Ah ! Pour les voyageurs ?

Interviewer : Oui.

P3 : Ah ! ils ne peuvent pas faire le bon de commande, ça ce n'est pas possible, mais...

Interviewer : Mais ça peut être utile ? S'ils pouvaient le faire ?

P3 : Oui, mais ce n'est pas possible. Parce qu'il y a le compte, ils ne peuvent pas le faire. Han...Oui ça pourrait être utile s'ils pouvaient le faire en direct.

Interviewer : Oui ?

P3 : Oui, bien sûr ! Ça nous ferait gagner du temps.

Interviewer : D'accord !

P3 : Mais... Oui, ça sera bien si vous pouvez faire leur demande de devis, et une fois que vous avez, vous, le devis, par exemple, vous avez le montant, vous connaissez le numéro de compte, sur lequel vous allez prendre la mission, vous tapez le numéro de compte, si vous savez que c'est montant-là va se déduire de ce compte

Interviewer : Et c'est moi-même que gère le budget, donc...

P3 : Voilà ! Mais, par contre, il faut que nous, nous soyons informés, quand même. Ça pourrait être bien. Mais je ne sais pas comment ça peut être réalisable. Mais ça pourrait être...ça résoudre le problème du vendredi, de la demande de devis du vendredi, par exemple.

Interviewer : D'autres suggestions ?

P3 : Non !

Interviewer : Pourriez-vous lister 3 fonctionnalités que vous aimeriez garder pour ce type de système ? •

P3 : Alors ! Trois ?

Interviewer : Oui !

P3 : Le suivi de vol qu'on a pris. Ça c'est bien, toute la liste des vols en cours et qu'on a été autorisé. Ça c'est important pour moi. L'effet de taper la référence aussi, en direct, ce n'est pas mal. Puisqu'il reçoit tout le suivi du billet.

Interviewer : La référence ?

P3 : La référence de réservation. Après la troisième...on est obligé d'élire trois ?

Interviewer : Non ! S'il n'y a pas trois, non ! Vous pensez que c'est juste les deux qui sont plus importants pour vous ?

P3 : Non. Puis, avoir un panel large de vol, ça c'est important, parce que...

Interviewer : Panel large de vols, ça veut dire, couvrir plusieurs compagnies aériennes ?

P3 : Oui ! Plusieurs compagnies et plusieurs horaires différents.

Interviewer : Pourriez-vous lister trois fonctionnalités que vous aimeriez changer pour ce type de système ?

P3 : Il y a une que m'angoisse. Quand on va sur le site, on tombe systématiquement sur le train. Et on prend plus de billet d'avion que de train. Et si on ne fait pas attention dans la précipitation on fait une demande d'aller et retour et puis on tombe sur la SNCF et pas sur...donc, on est obligé de changer d'onglet... je ne sais pas.

Interviewer : D'accord.

P3 : Ça c'est pénible...après... c'est quoi la question ?

Interviewer : Les trois fonctionnalités que vous aimiez changer pour ce type de système ?

P3 : J'aimerais qu'ils nous envoient les billets « *EasyJet* », beaucoup plus vite, parce que...ça...

Interviewer : Juste pour « *EasyJet* » ?

P3 : Oui, les autres...les autres aussi, mais « *EasyJet* », c'est beaucoup plus long, donc...

Interviewer : D'accord. Une troisième peut-être ?

P3 : Troisième ? Non, je ne vois pas.

Interviewer : Non ?

Section C : Rédaction des Récits Utilisateurs.

Interviewer : Bon ! Dans cette troisième partie on va évaluer un modèle pour décrire le processus besoin d'utilisateur, quand il utilise le système en fait. Donc, on est intéressé d'évaluer si ce modèle marche bien, ou s'il ne marche pas bien, si vous jugez que c'est un modèle qu'on pourrait utiliser, peut-être, et bon. Le modèle est plutôt comme ça.

On a l'histoire, un récit utilisateur, avec un titre, d'accord ? Avec un préambule, qu'on identifie en tant que le rôle, particulier...je veux faire quelque chose, je veux une fonctionnalité, afin de pouvoir obtenir un bénéfice, un but etc, etc...

Donc, pour ce récit-là, on a plusieurs scénarios. D'accord ?

P3 : D'accord !

Interviewer : Donc ; on a le scénario 1.

En tant que : donner un contexte, ou, plusieurs contextes, qu'on fait quelque chose, qu'on a un événement, alors, il y a un résultat ou plusieurs résultats.

P3 : D'accord !

Interviewer : Par exemple.

En tant que voyageur fréquent, je veux rechercher des billets, en fournissant des emplacements et des dates, afin de pouvoir obtenir des informations sur les tarifs et les horaires des vols. Donc, un scénario possible, c'est une recherche de ticket 'aller-simple'. D'accord ?

P3 : D'accord !

Interviewer : Par exemple :

En tant donné que je vais à la page "Recherche de vols"

Quand je choisis : "aller simple"

Et je tape "Paris" et choisis "Paris, Charles de Gaulle (CDG)" dans le champ "Départ de"

Et je tape "Toulouse" et choisis "Toulouse, Blagnac (TLS)" dans le champ "Arrivée à"

Et je choisis "2" dans le champ "Nombre total de passagers"

Et je choisis "15/12/2017" dans le champ "Date de départ"

Et je clique sur "Recherche"

Alors le système va afficher la liste des vols disponibles.

Donc, comme ça on fait une description de l'activité qu'on doit faire pour obtenir c'est but-là. D'accord ?

P3 : Oui !

Interviewer : Donc, c'est ça le modèle, on peut, bien sûr, décrire toutes les fonctionnalités, toutes les activités que l'utilisateur, il demande de faire sur le système, et quelle sera la réponse du système à cette demande-là. D'accord ?

P3 : D'accord !

Interviewer : Donc, ça c'est le modèle. Est-ce que vous pouvez écrire pour nous un exemple d'une situation, d'un problème, d'une demande utilisateur que vous identifiez, que vous recevez, par exemple, souvent ou que vous avez reçu récemment. Dans ce modèle-là ?

P3 : Oui ! Je dois écrire ?

Interviewer : Oui, s'il vous plaît ! Vous pouvez prendre un exemple.

P3 : Ça va être la même figure, mais au lieu d'aller simple, c'est aller/retour.

Interviewer : Oui, c'est une option. Vous pouvez aussi faire un scénario comme ça, pour enregistrer les données de passeport, pour enregistrer le bon de commande, pour confirmer la réservation, bon vous pouvez imaginer quelque scénario que vous voulez.

P3 : Alors, attendez !

Interviewer : Donc, on a toujours un contexte, d'accord ?

P3 : Oui !

Interviewer : Quand le scénario, il arrive, on a toujours une réponse. D'accord ?

P3 : D'accord !

Interviewer : Donc, on fait une action sur le système et on reçoit une réponse. D'accord ? En accord avec cette action qu'on a faite.

Donc, on va avoir ce qui l'utilisateur, qu'est-ce qu'il va faire, avec quel but. Et après on va avoir plusieurs scénarios, vous pouvez d'écrire un, par exemple, avec un contexte, une action et une réponse.

P3 : Mais, ça sera toujours dans le but d'une recherche d'un billet d'avion.

Interviewer : Ou après le rechercher. Faire la réservation, mettre les données ou mettre le bon de commande, bon voilà. Ça pourrait être fait après la recherche, après qu'on a déjà la liste, parce que, bon, on a la liste de vol disponible, après ça, qu'est-ce qu'on pouvait faire, par exemple, donc, mais, bien sûr, si vous voulez faire un scénario recherche, il n'y a pas de soucis. C'est juste pour clarifier que vous pouvez utiliser ce type de modèle pour décrire, n'importe quelle action sur le système.

P3 : D'accord !

Interviewer : C'est possible à vous de faire un exemple que vous plaît ?

P3 : Mais, je suis obligé de l'écrire ?

Interviewer : Oui, s'il vous plaît. Ce n'est pas forcément une obligation, mais bon.

P3 : À ce moment elle écrit. Après elle lit rapidement.

Interviewer : Mais dans c'est cas-là, comment le système, il donne la réponse ? C'est quoi qu'il montre ? Par exemple.

P3 : Si c'est autorisé, il écrit : Autorisé.

Interviewer : C'est un champ ou c'est écrire autorisé ?

P3 : C'est un petit onglet, qui est écrit : Autorisé. D'accord ?

Interviewer : D'accord ! Vous pouvez juste faire cette addition-là ? C'est comme le système donne la réponse.

P3 : À c'est moment-là, elle écrit.

Interviewer : C'est parfait ! C'est très bien, c'est exactement qu'on veut. Donc, pensez-vous pouvoir rédiger une liste de demandes/problèmes que vous recevrez au cours de cette semaine, c'est-à-dire, d'ici à mardi prochain ? Sur les problèmes que les utilisateurs vont rencontrer lors de la réservation de leurs voyages et qui va vous appeler pour résoudre ? Une liste simple.

P3 : Mais, si je n'ai pas un problème ?

Interviewer : Bon, s'il n'y a pas de demande que vous considérez comme important pour la réservation de voyage, même si ce n'est pas forcément un problème...

P3 : Si c'est une demande spécifique ?

Interviewer : Oui, une demande que vous considérez que c'est important, que c'est ...

P3 : D'accord !

Interviewer : Ou même la demande que vous ayez l'habitude de recevoir dans cette semaine. Donc, on a besoin d'une semaine de « *delay* ». Donc, d'ici à mardi prochain. Si vous pouvez, bien sûr. Enregistrer cette liste de demande, de problème, de demande ou de problème. D'accord ?

P3 : D'accord.

Interviewer : En qui concerne la réservation de voyage de manière intéressante et bonne. En faisant ça notre but est d'évaluer l'écrit de ce type d'histoire, donc, si possible, on vous demande pour chaque problème, que vous identifiez ou pour chaque demande que vous puissiez écrire cette demande aussi sur ce format-là. Vous pensez que c'est possible ? Donc, si, par exemple, ça a été arrivée hier, vous notez que ce une demande que vous ayez reçue et à côté vous ferez un exemple, en utilisant ce type de format. Vous pensez que c'est possible ?

P3 : Je ne sais pas. Je peux essayer de vous faire, mais je ne peux pas vous certifier.

Interviewer : C'est que nous intéresse, c'est plutôt d'avoir l'utilisateur comme vous, qu'écrivez ses histoires, pour qu'on puisse évaluer l'effectivité de ce type d'histoire dans une spécification de vision utilisateur. Donc, bon, si vous pouvez envoyer quelque exemple, que vous jugez simple, mais qui vont pouvez nous aider, ça sera bien.

P3 : Je vais essayer.

Interviewer : Je veux vous envoyer par mail aussi. Bon, merci beaucoup par vous aide.

4. TRANSCRIPTION : Participant 4 (P4)

Partie I : Questionnaire démographique et de contexte.

Interviewer : Bon, donc, la première partie, concerne un questionnaire démographique. Bon votre sexe.

P4 : Masculin.

Interviewer : Votre âge ?

P4 : 25 ans.

Interviewer : 25 ? D'accord ! Votre niveau d'étude ?

P4 : Bac

Interviewer : Depuis combien de temps, vous êtes au service de mission de l'IRIT ? Ça fait un mois, que vous avez me dit ?

P4 : Un moi, tout justement.

Interviewer : D'accord. Avez-vous déjà eu des expériences dans de services similaires ?

P4 : Oui

Interviewer : Pendant combien du temps ?

P4 : 04 ans à INSA de Toulouse.

Interviewer : Bon, pourriez-vous nous donner un aperçu de ce travail, en fournissant une brève description de vos tâches ?

P4 : D'accord ! Alors, on reçoit une demande d'auto mission des chercheurs. Donc, avec une demande de déplacement de voyage et avec cette demande-là, nous, on va sur le portail et puis on réserve, donc, le déplacement ou le train.

Interviewer : D'accord et après ? Ça finit avec la réservation ?

P4 : Voilà, nous, on fait la réservation et puis, après on reçoit une facture qui est directement payée avec nos services. On gagne la copie de la facture, mais c'est directement traiter par logiciel.

Interviewer : D'accord. Et la facture est envoyée par l'agence de voyages ?

P4 : Voilà, c'est pour courriel, du coup par l'agence de voyages.

Interviewer : D'accord

Partie II : Processus de traitement de demandes

Nous sommes intéressés par les préférences et les difficultés que les voyageurs de l'IRIT ont rencontrées et vous ont signalés lorsqu'ils essaient de réserver leurs voyages d'affaires. Nous sommes également intéressés par votre opinion sur les demandes reçues.

Label : * faits, • interprétation

Section A : Réception de demandes de réservation

Interviewer : Bon, concernant le processus de traitement de demandes. Comment les demandes de réservation des voyageurs arrivent-elles à vous et avec quelle fréquence ? Avez-vous des suggestions pour faire mieux ? *

P4 : Donc, du coup, via logiciel. Ils peuvent le faire directement sur le logiciel, où, on, nous mettons en validateur. Nous, on doit recevoir un mail pour valider le voyage ou sinon, on fait directement ou, ils nous passent des informations directement par mail, les chercheurs, et c'est nous qui réservons directement avec leur nom, prénom et le voyage.

Interviewer : Donc, d'accord. L'arrivée est directement via logiciel ou par mail ?

P4 : Voilà ! Ou sinon par mail avec toutes les infos.

Interviewer : D'accord ! Et avec quelle fréquence ?

P4 : Quelle fréquence ça peut dépendre.

Interviewer : Oui, en moyenne ?

P4 : En moyenne, dans le mois, je pense, 10 déplacements.

Interviewer : 10 déplacements par semaine ?

P4 : Non, par mois.

Interviewer : Par mois ?

P4 : Par mois, après, ça peut dépendre, comme des équipes que voyage beaucoup, alors, d'autres équipes...

Interviewer : qui ne voyagent pas de tout !

P4 : Et...voilà ! Donc, ça peut dépendre, on peut avoir 10, comme on peut avoir une vingtaine, ça dépend s'il y a des invités, s'il y a... et tout ça peut vraiment...je pense une dizaine par mois.

Interviewer : C'est la moyenne ?

P4 : Voilà, c'est la moyenne. Je pense !

Interviewer : D'accord. Et avez-vous de suggestion concernant ce sujet-là ? L'arrivée de demande ?

P4 : L'arrivée de demande ?

Interviewer : Oui, D'améliorer le processus. Avez-vous de suggestion ?

P4 : Moi, je trouve que ça marche vraiment bien. Quand le chercheur faisait directement la demande sur le logiciel. Comme ça, on a toutes les infos et puis, on a juste à faire le bon de commande, pour avoir un budget serré. Si c'est directement fait ou sinon, avec toutes les infos, ça nous fait perdre plus du temps, mais après, pour améliorer, je pense que les chercheurs peuvent aller sur le site pour réserver et en temps qu'on met comme validateur, ça va vite.

Interviewer : Combien de demandes de réservation de voyage avez-vous reçues la semaine dernière ? *

P4 : La semaine dernière, la demande de voyage, je n'ai eu quatre. Je n'ai pas beaucoup à ce moment. Comme je suis en train d'arriver...donc.

Interviewer : D'accord ! Bon, avez-vous des suggestions à ce sujet-là ? Bon, je pense que sur l'arrivée de demande, ce que le voyager fasse directement sur le système. *

P4 : Voilà ! Directement, c'est plus simple pour nous et je pense que pour eux, aussi. Je pense. Comme ça, il fait directement la demande.

Interviewer : D'accord ! Pensez-vous qu'il manque quelque chose dans la description des demandes de réservation que vous recevez ? •

P4 : Dans la description ? Non, je pense que c'est à ce clair.

Interviewer : Ce arrive assez complet ?

P4 : Normalement, c'est complet, après, voilà, si ce n'est pas fait directement par logiciel et que c'est le chercheur qui va nous envoyer, il peut avoir des modifications après, car ils ne sont pas sûrs, ils ne sont pas certains des horaires, mais, voilà. Si c'est directement par logiciel, au moins, c'est très bien, son vol, normalement. Si c'est fait directement. Les horaires, la description de vol entier.

Interviewer : Pensez-vous que c'est assez complet ?

P4 : Ah, oui ! C'est assez complet.

Interviewer : Il y a, par contre, de suggestion, pour faire mieux ? C'est...

P4 : Pour faire mieux ?

Interviewer : Dans la description spécifique...

P4 : Dans la description ?

Interviewer : Oui, de demande.

P4 : En plus, je ne pas sûr. Car je suis ici ne pas longtemps, donc...

Interviewer : Oui, c'est avec...Mais c'est juste ça qu'on va voir la différence.

P4 : C'est assez compliqué. Moi, pour l'instant. Moi, j'ai pu eu de complication vraiment. Mais là, je ne vois pas. Non, directement.

Interviewer : D'accord ! Est-ce que vous devez prendre des notes, par exemple : un post-it, dehors le système, sur les demandes de réservations que vous recevez ? *

P4 : Prendre de note...

Interviewer : De notes informelles, de choses...

P4 : Voilà ! Les numéros de vol, voilà, que demandent souvent les chercheurs. Le numéro de vol, puis et...après qui est que j'ai eu aussi ? Ah ! Et la gare aussi. Parce qu'à Lion, par exemple, j'ai eu un problème aussi avec Lion. Où on a plusieurs gares, à Lion...

Interviewer : Oui.

P4 : Et on ne sait pas forcément, le chercheur, dans quelle gare il va partir.

Interviewer : D'accord !

P4 : Donc, c'est sur tout ça, aussi. Et puis, après les horaires. Voilà.

Interviewer : D'accord.

P4 : C'est un complément d'information qu'on peut donner, du coup, pour le chercheur.

Interviewer : D'accord ! Et vous notez comment ?

P4 : Sur post-it.

Interviewer : Post-it ?

P4 : Sur post-it.

Interviewer : D'accord, et combien de notes par demande avez-vous dire normalement, vous prenez ? *

P4 : Généralement...combien de notes ? Je dirais deux.

Interviewer : Deux ?

P4 : Oui.

Interviewer : D'accord.

P4 : Après ça, c'est rapide, parce que c'est noté sur post-it, donc, du coup...

Interviewer : Bien sûr. Comment cela pourrait être meilleur ? La prise de notes au-dehors de système ou non ? Il faut noter tout sur le système. Vous imaginez quelle façon de faire ça, de manière plus productive ? Une partie du système pour faire ça, par exemple ?

P4 : Peut-être une partie du système, peut-être qui mettait en place une casse avec le numéro de vol, peut-être, ou même la gare, pour la gare. Pour le cas de Lion, préciser exactement quelle gare...

Interviewer : Ça n'existe pas dans le système ?

P4 : Non, non, non. C'est n'existe pas.

Interviewer : D'accord. Vous parlez du système le *Travel agent* ou GLPI ?

P4 : Alors. Sur le GLPI il n'y a pas.

Interviewer : Sur GLPI n'y a pas ?

P4 : Non, non.

Interviewer : Et sur le *Travel* ?

P4 : Après, là...

Interviewer : Non ?

P4 : Je ne sais pas de tout.

Interviewer : D'accord ! Vous n'avez pas encore utilisé ?

P4 : Non, non, du coup, non. Donc, c'est pour ça. Je ne pas encore toutes les autorisations, donc...

Interviewer : D'accord. Savez-vous me dire le nombre de notes que vous avez prises la semaine dernière, pour la demande ?

P4 : Là, j'ai pris beaucoup. Parce que comme je suis en train de commencer, j'ai pris beaucoup de notes sur le post-it.

Interviewer : D'accord. Savez-vous quantifier ? Peut-être ?

P4 : Là, je suis en train d'apprendre, au moins une vingtaine.

Interviewer : Au moins une vingtaine ?

P4 : Une vingtaine, parce que j'ai eu un gros déplacement de plusieurs personnes et...huit personnes, donc, pour tout gérer, j'ai tout noté, parce que, comme je suis en train de commencer, je ne voulais pas, non plus, manquer un truc. Donc, à ce moment j'ai pris beaucoup.

Interviewer : D'accord.

P4 : Mais généralement...généralement, je ne prends pas énormément.

Interviewer : D'accord ! Bon. Si vous prenez des notes, c'est le cas, comment vous les conservées et sur quel format ? Donc, c'est sur le post-it, c'est ça ? *

P4 : Post-it, voilà.

Interviewer : Et la suggestion d'amélioration... c'est de pouvoir...

P4 : C'est de directement sur le logiciel, avoir une casse, ou un...

Interviewer : Un post-it virtuel ? Peut-être ?

P4 : Voilà, ou un complément, voilà.

Interviewer : D'accord.

P4 : Un complément avec toutes les notes qu'on ne peut pas mettre ailleurs, ça serait pas mal.

Interviewer : D'accord ! Pensez-vous que l'enregistrement de ces notes est important ? •

P4 : Ah ! Oui, oui.

Interviewer : Oui ?

P4 : Oui, oui.

Interviewer : Pourquoi ? Et comment vous pouvez améliorer ça ?

P4 : Ah ! Pourquoi et comment ? Dans la demande de l'ordre de mission, avoir un complément, une casse complément avec vraiment toutes les informations. Comme ça, nous évitons de nous reprendre de notes après.

Interviewer : D'accord. Et vous croyez que c'est important pourquoi ?

P4 : Moi, je sais que ça me sert beaucoup, parce que, comme ça, je vraiment toutes les informations que sont claires. Je n'ai pas besoin de retourner à la réservation, pour avoir ces informations. Donc, pour moi je noterai tout ça sur le papier et puis pouvoir faire des post-it.

Interviewer : D'accord. Pouvez-vous fournir quelques exemples de demande de réservation que vous recevez ? *

P4 : Comment ?

Interviewer : Pouvez-vous fournir quelques exemples de demande de réservation que vous recevez ?

P4 : Demande ?

Interviewer : Oui ! De réservation de voyage. C'est quel contenu normalement ? Il y a de...

P4 : Alors, il y a de date, des horaires. Quoi d'autre ?! La destination, bien sûr, et je pense que c'est tout.

Interviewer : Le moyen de transport ?

P4 : Et le moyen de transport.

Interviewer : D'accord ! Il vient avec des données des passagers déjà, oui ou non ?

P4 : Avec le... ?

Interviewer : Avec les données des passagers. Le nom, la date de naissance.

P4 : Ah ! Oui, bien sûr ! Parce que nous, on a quand même une fiche avec la demande d'ordre de mission.

Interviewer : D'accord.

P4 : Donc, il y a déjà le nom, si c'est un chercheur, s'il est de notre... s'il est chez nous ou s'il est d'ailleurs.

Interviewer : D'accord !

P4 : Là, pour ça, c'est vraiment rempli, donc...

Interviewer : D'accord.

Section B : Traitement des demandes de réservation.

Interviewer : Bon, concernant le traitement de demandes de réservation. Quelle est la procédure-type de traitement de réservation ? On commence, pourquoi ? Après qu'est-ce que vient ? *

P4 : Nous, on a déjà la demande d'ordre de mission qui est déjà rempli par le chercheur.

Interviewer : D'accord.

P4 : Avec ça, nous, on fait l'ordre de mission, l'OM. Donc, voilà. Après nous, on fait, la réservation de voyage, donc, le bon de commande et depuis le numéro d'ordre de mission qu'on renseigne sur le bon de commande de voyage et voilà. Et après, le chercheur part en mission et juste après sa mission, c'est le

retour de la mission et il revient avec toutes les pièces justificatives que nous, après, on traite et qu'on remplit l'état de frais, qu'on envoie, du coup à l'agence comptable pour le remboursement.

Interviewer : D'accord ! Vous avez de suggestion dans ce processus-là, pour améliorer ?

P4 : C'est peut-être avec les pièces justificatives, après je ne vois pas comment on peut améliorer, mais...

Interviewer : Pour quoi ? Qu'est-ce qu'arrive la pièce ?

P4 : Parce que, parfois, il y a vraiment de gros paquets de pièces de métro. Bah ! S'il peut avoir une fiche que renseignant toutes les pièces, parce que, en plus, on peut le perdre aussi.

Interviewer : D'accord.

P4 : Si, on peut le faire une fiche avec le nombre de tickets de métro, le nombre de tickets de bus et avec, directement le prix qui sont à côté, parce que, parfois, on cherche les tickets métro, on cherche le prix et voilà...

Interviewer : D'accord, mais ça c'est après le voyage.

P4 : Voilà !

Interviewer : Après le voyage ?

P4 : C'est le retour.

Interviewer : D'accord. C'est pour faire...

P4 : L'état de frais.

Interviewer : D'accord.

P4 : Voilà.

Interviewer : Très bien ! Est-ce que dans les demandes des voyageurs que vous traitez il y a des informations qu'on pourrait identifier comme des besoins et/ou des exigences pour améliorer un système de réservation de voyage ? •

P4 : Pour l'instant je n'ai eu un, mais ça pourrait arriver, oui.

Interviewer : Oui ?

P4 : Après... Qu'est-ce qu'on peut avoir ? Après, je ne vois pas, parce que je suis en train de commencer. J'essaye de repenser à un cas que j'ai eu.

Interviewer : Dans le cas que vous recevez récemment, il y a...

P4 : Il n'y a pas d'exigence particulière, mais...c'est depuis que j'ai commencé ici, non ? Ça ne me pas arrivée encore. Ma j'essaie de me rappeler avant.

Interviewer : Mais tout que vous avez reçu, vous avez bien réussi pour le faire à le chercher sur le système, par exemple ?

P4 : Oui, oui, Bien sûr. Oui, oui.

Interviewer : D'accord.

P4 : Après le chercheur, nous, on dit aussi. Donc, s'il y a vraiment de choses particulières. Donc, il va nous dire, mais c'est vrai qui là, par instante, moi, ça n'est pas encore arrivée.

Interviewer : **Quelles sont vos besoins/exigences pour le système de réservation de voyage que vous utilisez actuellement ? ***

P4 : Des exigences ?

Interviewer : Qu'est-ce que vous considérez comme important et que vous pensez de c'est utile ?

P4 : Pour moi, ça me plaît. Je ne sais pas. Je pense que nous, on peut améliorer en fait.

Interviewer : D'accord.

P4 : Juste la demande qui doit faire directement par le chercheur. Ça nous avance beaucoup.

Interviewer : D'accord.

P4 : Et après...non. Parce que c'est vraiment vit, si on a toutes les infos. Il n'y a pas à améliorer. Bon, je ne vois pas.

Interviewer : **D'accord ! Avec quelle fréquence vous devez demander l'aide des autres membres de l'équipe pour résoudre les demandes ? ***

P4 : Bon, du coup, beaucoup. Mais quand j'ai travaillé à INSA Toulouse, pendant quatre ans. C'est que j'ai fait aussi. Et, non, ça ne m'arrivait pas. Alors, quand on prend vraiment de cas particulier, où on vraiment demande de l'aide, parce que, mais...pendant le mois, deux fois.

Interviewer : Deux fois ?

P4 : Deux fois dans le mois.

Interviewer : D'accord.

P4 : Après, c'est vrai que j'ai beaucoup demandé, mais...

Interviewer : D'accord. Comment on peut améliorer ça. Il y a une façon de ...

P4 : Alors, d'améliorer...Après c'est vrai que pour nous on est dans un « *open space* » donc, on dialogue directement tous ensemble. Mais, c'est vrai que, quand, je travaillais aussi à Marseille et j'étais tout seul dans mon bureau et c'est vrai, que, quand il avait de cas comme ça, que m'arrivais, c'était un peu compliqué, parce que je travaillais un peu plus par mail, où je n'avais pas forcément la réponse directement. Et c'est vrai que là, du coup, quand on est tous à côté, ça facilite beaucoup.

Interviewer : D'accord.

P4 : Pour moi, le problème est directement résolu. Voilà.

Interviewer : D'accord. Mais, aujourd'hui, même avec ce bureau ouvert, il y a de suggestion que vous pensez que c'est utile ? Améliore encore plus ?

P4 : Encore plus ? Pour le cas comme ça ?! Peut-être avoir de fournisseur aussi. Si on a de cas particulier sur le voyage qui sont à l'étranger, qui sont un peu compliqués, peut-être avoir une aide de fournisseur, du coup.

Interviewer : C'est l'agent de voyage ?

P4 : Voilà ! du coup, voilà. Peut-être une petite note, une procédure pour de cas particulier, comme ça que sort, comme à mettre au jour.

Interviewer : D'accord. Avec quelle fréquence vous devez demander aux voyageurs de clarifier les informations concernant leur demande ? *

P4 : Beaucoup de fois. Beaucoup de foi oui.

Interviewer : Et comment on pourrait améliorer ça ?

P4 : Bah ! C'est assez un peu compliqué. Parce que, c'est un peu le chercheur de tout nous dire, du coup, parce que, c'est vrai que nous, à chaque fois on revient vers eux, on alors le redemande et puis. Et c'est pour ça que, quand si est fait directement sur le logiciel, nous, on avait juste à valider et puis, c'est directement fait.

Interviewer : D'accord.

P4 : Donc, du coup, sinon, on est obligé de courir après les informations et...

Interviewer : D'accord, très bien. Selon votre propre expérience, quelles seraient les fonctionnalités qui vous seraient utiles et qui devraient être rajoutées au logiciel de réservation ? Qui n'est pas là aujourd'hui, par exemple. *

P4 : Ah ! C'est peut-être de garder les voyages qu'on a réservés.

Interviewer : Ça n'existe pas ?

P4 : Non, parce qu'il s'élève directement, après la date de retour le voyage part et on ne sait pas, du coup...Et c'est vrai que la dernière fois, j'ai cherché...j'ai cherché un voyage, en plus, quand j'étais ici et c'est à éléver du module de réservation. J'ai dû retourner sur l'OM pour avoir exactement les dates, les horaires de vols et tout ça.

Interviewer : D'accord. Vous parlez de quel système ici exactement ? De GLPI ?

P4 : SIMBAD

Interviewer : SIMBAD ! D'accord !

P4 : De que le voyage est passé, il s'élevait directement de la base.

Interviewer : Quelque chose de plus ?

P4 : Non, après je ne crois pas. Non !

Interviewer : Non ?

P4 : Non !

Interviewer : Selon votre propre expérience, quelles seraient les fonctionnalités qui seraient utile pour les voyageurs ?

P4 : Pour les voyageurs ?

Interviewer : Oui, pour les chercheurs.

P4 : Pour les chercheurs, moi je dirais plutôt pour...pour nous, pour les gestionnaires avoir vraiment à qui demander la validation.

Interviewer : À qui ? C'est ça ?

P4 : Voilà, c'est ça. Parce que nous on gère tous les budgets des équipes et c'est que les chercheurs viennent dans le bureau juste pour demander à qui il doit mettre valideur.

Interviewer : D'accord. Donc, ils ne savaient pas ça normalement ?

P4 : Pas forcément, parce que, vu que moi, je suis nouveau, donc, j'ai la peine d'arriver, donc, c'est vrai ils ne savaient pas trop, parce qu'on a changé, du coup, les équipes, et, donc, ils ne savaient pas trop, si c'était moi, si c'est une autre gestionnaire. Voilà.

Interviewer : D'accord !

P4 : Un petit rappelle sur ça, ça les évite de rentrer dans le bureau juste pour demander, une information comme ça.

Interviewer : D'accord ! Quelque chose en plus ?

P4 : Non, non. Après je ne vois pas.

Interviewer : Pourriez-vous lister 3 fonctionnalités que vous aimeriez garder pour ce type de système de réservation de voyage ? •

P4 : Que je voudrais garder ?

Interviewer : Oui. Quels sont les plus utiles ? Que vous pensez être indispensable à votre avis.

P4 : Je ne sais pas de tout.

Interviewer : La partie de chercher de vol. La partie d'ordonner par data, la partie de renseigne des infos. Voilà. Quelque chose que vous considérez, le trois, que vous considérez plus importants.

P4 : Après, c'est vrai que le site c'est fluide, donc, il va vachement vite. Après il est assez complet aussi, parce qu'on a tous les vols, avec tout. Je ne sais pas comme en dire. En fait, il y a beaucoup d'informations qui est sur le site. Quoi je peux dire de plus ?

Interviewer : Mais, par exemple, en termes de fonctionnalité vous considérez que la partie plus importante c'est pouvoir chercher dans plusieurs compagnies. Pouvoir ordonner par prix, par exemple. Pouvoir renseigner les infos directement par un autre système, parce que les données sont déjà dans la base. Bon, voilà ! Quelque chose que vous considérez...

P4 : Voilà ! C'est... moi. Je préfère, parce que nous, on renseigne toutes les infos à la fin. Quand on a réservé le vol et c'est vrai que c'est complet, comme ça on a tout directement et c'est nous qui renseignons. Après, il s'est assez compliqué, parce que, oui, on a pleine de compagnie. Je pense que s'est classé par horaires, mais aussi, on peut chercher aussi par tarif, c'est qui est pas mal aussi. Puis, on a toutes les infos à la fin que c'est bien aussi et la validation est aussi simple. Une fois qu'on a réservé le vol pour pouvoir valider, on a juste copié le numéro de bon de commande et puis comme ça, c'est validé.

Interviewer : D'accord ! Pourriez-vous lister trois fonctionnalités que vous aimeriez changer pour ce type de système ? •

P4 : Changer ? Après c'est totalement... je ne sais pas, Non ! Changer je ne vois pas, je ne vois pas en fait. Parce que ce n'est pas totalement sur le logiciel. Le logiciel est vraiment bien.

Interviewer : Vous ne changiez rien ?

P4 : Non ! Je ne pense pas. Je ne pense pas !

Section C : Rédaction des Récits Utilisateurs.

Interviewer : Bon ! Dans cette partie on va évaluer le modèle, qu'on a prescrit pour décrire les actions d'utilisateur et les réponses que le système donne sur ses actions. Donc, on appelle ça : le récit utilisateur. Et, on suit un modèle d'écrire ça. Donc, dans le modèle on a un préambule, d'accord ?

P4 : D'accord !

Interviewer : Donc ;

En tant que [rôle ou personne]

Je veux [fonctionnalité]

Afin de [but, bénéfice ou valeur de la fonctionnalité]

D'accord ?

P4 : D'accord !

Interviewer : Bon, et pour c'est préambule-là on a plus de scénarios possibles. Donc, chaque scénario, on a une description de scénario, un contexte qu'on donne par la clause « **En tant donné** » ; ou plusieurs contextes. On peut ajouter plusieurs contextes, ou une action, un évènement, et le résultat que le système va nous donner. Après cet évènement, on registre ça dans la clause « **Alors** ».

P4 : D'accord !

Interviewer : Donc, comme exemple :

En tant que voyageur fréquent,

Je veux rechercher des billets, en fournissant des emplacements et des dates,

Afin de pouvoir obtenir des informations sur les tarifs et les horaires des vols.

D'accord ?

P4 : D'accord !

Interviewer : Donc, dans cette histoire, scénario possible ; une recherche de ticket aller simple, par exemple. Donc,

En tant donné que je vais à la page “Recherche de vols”

Quand je choisis : “aller simple”

Et je tape “Paris” et choisis “Paris, Charles de Gaulle (CDG)” dans le champ “Départ de”

Et je tape “Toulouse” et choisis “Toulouse, Blagnac (TLS)” dans le champ “Arrivée à”

Et je choisis “2” dans le champ “Nombre total de passagers”

Et je choisis “15/12/2017” dans le champ “Date de départ”

Et je clique sur “Recherche”

Alors le système va afficher la liste des vols disponibles.

C'est la réponse que le système va me donner suite à cette action-là. D'accord ?

P4 : D'accord !

Interviewer : Donc, c'est un modèle qu'on utilise pour décrire l'interaction entre l'utilisateur et le système, D'accord ?

Donc, en tant donné un scénario on va, quand certes évènement va arriver, le système va nous donner quelque réponse. D'accord ? Donc, le but c'est de prendre quelques exemples que vous avez eus à la semaine dernière ou bon, dès que vous êtes ici, et bon, je voudrais que vous fassiez un exemple pour moi, en utilisant ce modèle. D'accord ?

P4 : D'accord !

Interviewer : Vous pouvez choisir le contexte, bon, vous le décrivez, par exemple, la partie de renseigne le donné passager, vous voulez décrire une recherche du billet de train pour...je ne sais pas quoi, pour plusieurs destinations. Vous pouvez décrire, par exemple la partie validation. Je veux valider un billet et après, vous pouvez décrire un problème, une situation d'erreur, par exemple. Quand vous n'informez pas l'aéroport d'arrivée, qu'est-ce qu'arrive, quel type de sortie le système donne ? Bon, vous pouvez décrire qu'est-ce que vous voulez, d'accord ? Donc, vous prenez un exemple et s'il vous plaît, vous décrivez cet exemple, en utilisant ce modèle. Vous pensez que c'est possible ?

P4 : D'accord !

Interviewer : Je vous donne une feuille. Bon, je peux vous aider, si vous voulez faire ensemble, je peux vous aider, bien sûr.

P4 : Donc, du coup, je dois décrire une tâche.

Interviewer : Oui ! Sur le système de réservation de voyage.

P4 : D'accord ! Où je dois cliquer sur le module de la réservation.

Interviewer : Oui, oui. La première partie c'est définir qu'est-ce que vous voulez faire. Vous avez pensé à quelle tâche, par exemple ? Vous pouvez choisir.

P4 : D'accord ! La destination.

Interviewer : Non, non ! Une tâche complète, vous voulez faire, vous voulez donner, les données des passagers, par exemple. Dès qu'on a pris déjà la liste de vol disponible, par exemple, d'accord ? Vous voulez renseigner les données passagères. Ça c'est un exemple. Ça c'est une tâche.

P4 : D'accord !

Interviewer : Donc, un contexte ça sera la liste déjà disponible, par exemple, d'accord ? Je veux faire une action, c'est renseigner les données des passagers, les infos personnelles etc. et etc...et je peux décrire par exemple quel type de sorti, par exemple, le système pourra me donner, il va me montrer une page avec la confirmation des données, pour que je puisse confirmer que les données sont bon, par exemple.

P4 : D'accord !

Interviewer : Ça c'est une tâche, d'autre tâche, vous pouvez décrire commet vous faites un voyage par train pour plusieurs destinations. Alors je vais sur la page SNCF, je vais trouver la gare, etc., etc. Et le système va me donner le voyage en train disponible. D'autre situation, vous pouvez faire un enregistrement

de demande. Après que la voyage a été approuvé. Donc, le voyageur déjà envoyé l'itinéraire, tout ça, il déjà utilisé le système de voyage et vous allez juste mettre le numéro de bon de commande. C'est une option. Donc, la première partie c'est de choisir une tâche n'importe pas quelle tâche, d'accord ?

P4 : D'accord !

Interviewer : Que vous voulez décrire ?

P4 : D'accord !

Interviewer : Quel tâche vous pensez que c'est intéressant de décrire, comme exemple ?

P4 : La réservation ! Afin de la réservation, de prendre le billet de train ou de vol, avec tout le résultat, les horaires et puis sur une journée.

Interviewer : D'accord, mais dans un cas spécifique, par exemple. Sur c'est tâche-là. Donnez à moi un scénario. Moi, je suis un voyageur, je vais faire. Non, vous. Vous êtes un voyageur, vous voulez aller où ?

P4 : À Paris !

Interviewer : Oui ! Pour combien de temps ? Dans quelle date ?

P4 : Une semaine. Donc, du coup, je décrire comment je fais sur le...

Interviewer : Sur le système !

P4 : D'accord !

Interviewer : Oui !

P4 : Alors, je vais insérer mon nom et mon prénom, je vais sur la réservation, j'avance au lieu de départ et le lieu d'arrivée, les dates, les horaires, après je clique sur la recherche, je toute une liste, je choisis, donc, du coup, je choisis mon vol, avec le prix que m'intéresse, avec tout que m'intéresse. Une fois que je fais ça, je valide, j'ai une autre fenêtre que ça fiche avec mon aller et mon retour, qui sont bien pris à charge avec le prix, je réserve et puis je renseigne, du coup, mon nom et mon prénom, l'objet de mon départ et je choisis et je vais valider.

Interviewer : D'accord ! Très Bien ! Bon, vous pouvez décrire tout ça, vous pouvez décrire une partie. D'accord ?

P4 : D'accord !

Interviewer : Donc, l'importance que vous faites cette description de ce scénario, que vous avez me donner à toute l'heure, dans ce modèle-là. Donc, vous avez décrit un préambule, d'accord ?

P4 : D'accord.

Interviewer : En tant que...

Je veux...

Afin de...

Et vous pouvez décrire un ou plusieurs scénarios, dans ce contexte-là et pour chaque scénario, vous allez donner un contexte, une action et un résultat que le système va donner.

P4 : D'accord !

Interviewer : D'accord ?

P4 : Moi, je le fais maintenant ?

Interviewer : Oui, s'il vous plaît. Si vous pouvez.

P4 : Il écrit.

Interviewer : Vous êtes obligé juste à utiliser :

En tant que...

Je veux ...

Alors, etc.

P4 : Il écrit.

Donc, là le scénario. Et là j'ai choisi quelque chose que je vois.

Interviewer : Oui ! Vous avez donné un contexte...En tant que quelque chose arrive, quand je fais quelque action, alors le système va me montrer quelque chose.

P4 : D'accord ! Il écrit.

Interviewer : Si vous avez de doute, n'excitez pas.

P4 : Il écrit et lit en voix bas – Je pense que c'est bon.

Interviewer : Donc, en tant que je... Il lit en voix bas.

D'accord ! Très bien ! Donc, c'est que je vous demande c'est...d'ici à mardi prochain, donc, dans un « *delay* » d'une semaine. Est-ce que vous pouvez rédiger une liste de demande, de problème que vous allez recevoir ?

P4 : Oui, Bien Sûr !

Interviewer : Oui ? C'est une liste simple, donc, bon...j'ai eu une demande pour faire ça, pour faire ça, l'utilisateur il y a rencontré de problème pour chercher ça, ça, dans ce contexte-là, et par ce problème-là vous pouvez aussi le décrire sur cette forme-là.

P4 : D'accord !

Interviewer : Oui ? Vous pensez que c'est possible ?

P4 : Oui, bien sûr. Je peux garder la fiche ?

Interviewer : Oui, oui ! Bien sûr ! Je vais vous envoyer par mail aussi, avec un modèle, juste pour faciliter la tâche, et donc, d'ici à mardi prochain, donc, ça va faire une semaine. Si vous pouvez m'envoyer jusqu'à vendredi prochain. D'accord ?

P4 : Vendredi prochain ?

Interviewer : La fin de la semaine prochaine.

P4 : D'accord !

Interviewer : Ça sera bien. Par mail aussi.

P4 : Ok.

Interviewer : Donc, c'est une liste simple de problèmes de demandes que vous avez reçue et une autre liste de ces problèmes décrire de cette façon-là, en utilisant ce modèle.

P4 : Ok !

Interviewer : Vous pensez que c'est possible ?

P4 : Bien sûr !

Interviewer : Très bien ! Donc, je vous demande de me donner votre mail et après à la fin, je vais vous envoyer un petit questionnaire, aussi sur l'utilisation de ce modèle-là. Très bien P4 ! Merci beaucoup. Je vais vous envoyer le mail et je vais attendre votre retour la semaine prochaine.

P4 : La semaine prochaine.

Interviewer : D'accord !

P4 : Et du coup, ça sera pour vendredi.

Interviewer : Oui, le prochain.

P4 : Vendredi prochain.

Interviewer : Oui ! Parce qu'on va prendre une semaine de demande, donc, d'ici à mardi prochain et de mardi prochain à vendredi, vous pouvez m'envoyer le résultat.

P4 : D'accord ! Ok !

Interviewer : D'accord ! C'est bon ?

Annex C: Transcription of the Interviews

P4 : Oui ! Bien sûr !

Interviewer : Merci beaucoup, bon après-midi et avoir !

P4 : Merci, avoir !