Automated Support for Consistency Management and Validation of Requirements

Massila Kamalrudin

A thesis submitted in fulfilment of the requirements for the degree of Doctor of Philosophy of Electrical and Electronic Engineering, The University of Auckland, 2011.

Abstract

The requirements engineering phase of software development remains presents many challenges to researchers and practitioners alike. Among them, the management of consistency across multiple representations is particularly complex yet it lacks effective tool support. The thesis proposes an automated support mechanism to enable users (Requirements Engineers) to manage the consistency and validation of requirements. We have investigated existing approaches, developed a novel technique, and realised this technique as an automated support tool called MaramaAI (Automated Inconsistency Checker).

We have taken an an iterative approach to our work. We began by developing a lightweight extraction approach that allows an accurate and quick extraction of essential requirements (abstract interactions) from natural language requirements and the generation of Essential Use Case models from them. We then used automated traceability with visual support to check the consistency of requirements in three different forms: textual natural language requirements, abstract interaction and Essential Use Case, as well as to further validate the correctness and completeness of requirements. We also extended the automated tool to provide end-to-end rapid prototyping support embedded in the tool for validating requirements consistency in a form usable by both requirement engineers and clients to confirm the consistency of requirements.

We have evaluated the tool’s efficacy and performance especially on the extraction process, and also evaluated the user perception on the tool’s usability and user-perceived strengths via a substantial usability study and applied the tool to several case studies. The results were positive, and demonstrate that MaramaAI can be used to manage the consistency and validation of requirements in various domains of applications.

Dedication

I dedicate this thesis to my husband Ahmad Fadzil Mohamad, my father Kamalrudin Jantan and my mother Nya Lily Baba.

“Your endless Prayers, love and support have gotten me here”

Acknowledgement

Alhamdulillah: all praises to Allah for giving me good health, a strong will and a clear path to complete this study.

I am heartily thankful to my supervisors, Professor John Hosking and Professor John Grundy, for their guidance and support throughout my PhD study. For almost three years of collaboration, I have learned to think positively, to be confident and focused, to appreciate comments and criticisms, to take up challenges, to try hard to improve weaknesses, to be generous to others and to love and care for my family. I am really glad to have had both of you as my supervisors.

To John & John: You're really great! You have given me such a great time while going through this tough process!

My highest gratitude to my parents (Abah, Mak, Mama and Papa) for all their endless prayers, love, confidence and support to me to go through the ups and downs of this journey.

My warmest appreciation to my beloved husband, Ahmad Fadzil Mohamad, for all his sacrifices, understanding, love, care, motivation and support. Acting as my third supervisor at home, he has really shielded me from all the laziness and negative influences.

To my hubby: Yang, this PhD is yours!

I would like to thank all my friends, especially Jun Huh for his kindness and help throughout this study.

I would also like to thank Silicon Dream Ltd. for sharing their requirements for this research.

I would also like to acknowledge the sponsor of my study: Ministry of Higher Education and UTeM as well as PReSS Account of University of Auckland and the FRST Software Process and Product Improvement project.

Finally, I would like to share this quote which has always been with me for almost three years to keep up the high level of enthusiasm in completing my research:

"Forget mistakes. Forget failure. Forget everything except what you're going to do now and do it. Today is your lucky day." ***Will Durant***

Table of Contents

Abstract ii

Dedication iii

Acknowledgement iv

Table of Contents v

List of Figures x

List of Tables xv

Chapter 1: Introduction 1

1 Research Background 1

1.1 What is a Requirement? 1

1.1.1 Consistency 2

1.1.2 Inconsistency 3

1.2 Research Motivations 4

1.3 Research Questions 5

1.4 Research Objective 7

1.5 Research Methodology 7

1.6 Research Contributions 9

1.7 Thesis Organisation 11

1.8 Summary 12

2.1 Requirement Management 13

2.2 Requirement Modelling Technique 14

2.3 Requirements Specification 14

2.3.1 Formal Specification 14

2.3.2 Semi-formal Specification 15

2.3.3 Informal Specification 15

2.4 Semantics in Requirements 16

2.5 Requirement Engineering Tools (RE Tools) 16

2.5.1 Examples of Commercial Requirement Engineering Tools 16

2.5.2 Examples of Research Requirement Engineering Tools 19

2.5.3 Discussion of RE Tools Features 23

2.6 Requirements Validation 25

2.7 Consistency/ Inconsistency Management 27

2.7.1 Consistency/ Inconsistency Management Techniques in General 27

2.8 Related Work of Consistency/Inconsistency Management 29

2.8.1 Traceability 29

2.8.2 Analysis Approach 30

2.9 Analysis of Consistency / Inconsistency Management Research 39

2.10 Discussion 43

2.11 Summary 44

Chapter 3: Motivation and Overview of Our Approach 45

3.1 Introduction 45

3.2 Overview of Essential Use Cases (EUCs) 46

3.3 Applying Essential Use Cases: A Study 48

3.4 Overview of Our Approach 50

3.5 Summary 54

Chapter 4: Essential Interaction Extraction 55

4.1 Introduction 55

4.2 Essential Interaction Pattern Library 57

4.3 Tool Support 60
4.3.1 Tool Process 60
4.3.2 Tool Example 62
4.4 Evaluation 64
4.5 Summary 69
Chapter 5: Managing Requirements Consistency 71
5.1 Introduction 71
5.2 Managing the consistency 71
5.3 Tool Support 73
5.4 Architecture and Implementation 83
5.5 Evaluation 89
5.6 Summary 93
Chapter 6: Requirements Quality Checking 95
6.1 Introduction 95
6.2 EUC interaction pattern 96
6.3 Our Approach 97
6.4 Tool Support & Usage 100
6.4.1 Tool Support 100
6.4.2 Consistency Checking 101
6.4.3 Inconsistency, Incorrectness and Incompleteness Checking 105
6.5 Architecture and Implementation 109
6.6 Conclusion 117
Chapter 7: Supporting Requirement Validation via End-to-End Rapid Prototyping 118
7.1 Introduction 118

7.2 Background 119

7.2.1 Rapid Prototyping 119

7.2.2 Essential User Interface (EUI) prototyping 119

7.3 Applying EUI Rapid Prototyping: A Study 120

7.4 Related Work 123

7.5 Our Approach 125

7.6 EUI Pattern Library 127

7.7 EUI Pattern Template library 129

7.8 Tool Support 130

7.9 Architecture and Implementation 134

7.10 Evaluation 138

7.11 Summary 141

Chapter 8: Case Studies Examples 142

8.1 Introduction 142

8.2 Case Study 1: Reserve a Vehicle from a Rental Company 142

8.2.1 Example of Usage 143

8.3 Case Study 2: Book Check-out in a Library System 151

8.3.1 Example of Usage 152

8.4 Case Study 3: Manage Events with Event Listing System 160

8.4.1 Example of Usage 161

8.5 Discussion and Summary 169

Chapter 9: Evaluation 170

9.1 Evaluation Mechanism Overview 170

9.2 Usability Criteria for Usability Evaluation 171

9.3 Cognitive Dimensions of Notations Approach (CD) 172

9.4 Design of the Study 172

9.5 Survey Method 175

9.6 Survey Result and Analysis 177

9.7 Comparison of Survey Results 195

9.8 Summary 199

Chapter 10: Conclusion and Future Work 200

10.1 Summary of Research Contributions 200

10.2 Conclusions 201

10.3 Future Work 202

10.4 Summary 204

APPENDIX A: PARTICIPANT INFORMATION SHEET (HEAD OF DEPARTMENT)................. 205

APPENDIX B: PARTICIPANT INFORMATION SHEET (STUDENT) 207

APPENDIX C: CONSENT FORM (HEAD OF DEPARTMENT)... 210

APPENDIX D: CONSENT FORM (STUDENT).. 211

APPENDIX E: SURVEY QUESTIONNAIRES 212

REFERENCES 228

List of Figures

Figure 1.1: Requirement Analysis and Negotiation Process (From [5]) 2

Figure 2. 1: Example of Features in RaQuest [66] 17

Figure 2. 2: Example of the Qpack Tool for (1) Requirement tracking and (2) Requirement traceability [67] 18

Figure 2. 3: Example of the Enterprise Architect 8 tool in managing: (1) requirements (2) internal requirements and (3) exporting the internal requirements [3] 19

Figure 2. 4: Example of the EA-Miner tool in eliciting the input requirement(1) and identifying concerns (2) [9] 20

Figure 2. 5: Example of the LRS Requirement document (1), LRS scenario object and LRS task model (3) generated from the LRS tool [8] 21

Figure 2. 6: Example of the WikiReq system in eliciting and managing requirement: (1) shows the actor view point page, (2) goal view point page and (3) the WikiReq exported to Eclipse [12] 22

Figure 2. 7: Example of Results from the NAI tool in detecting nocuous coordination ambiguity [7] 23

Figure 2. 8: Heat Map Representations: Categorisation of the Type of Contributions, Techniques, Specifications and Semantics Used for Checking the Consistency/Inconsistency 40

Figure 2. 9: Heat Map representation: Classification of the Model Used as a Semi-Formal Specification Approaches 42

Figure 3.1 : (left) Example of Textual Natural Language Requirements and (right) Example of Essential Use Case (EUC model) [4] 47

Figure 3. 2: The scenario “Getting Cash” Refined and Adapted from [4] Used For the Evaluation 48

Figure 3. 3: Overview of Our Requirements Consistency with End-to-End Support using EUC and a Traceability Management Approach 52

Figure 4. 1: Our Essential interaction extraction approach 56

Figure 4. 2: Tree Structure for Key Textual Phrase 59

Figure 4. 4: Our Automated Tracing Tool 62

Figure 4. 5: The Tool Usefulness Results....66

Figure 4. 6: The tool Ease of Use Results....66

Figure 4. 7: Accuracy across different scenarios....68

Figure 5. 1: Framework for Extracting Requirement (1) Mapping interactions (2) and Creating the EUC Automatically (3)....73

Figure 5. 2: Tracing the Abstract Interaction from Textual Natural Language Requirement and Mapping to the Marama Essential....75

Figure 5. 3: Trace back from EUC diagram in Marama Essential to the Abstract Interaction and Textual natural language requirement....76

Figure 5. 4: Inconsistency Occurring: Change of Sequence of Abstract Interaction....77

Figure 5. 5: Inconsistency Occurring: Change of Sequence of EUC component....78

Figure 5. 6: Inconsistency Occurring: Adding New Item to the Abstract Interaction....79

Figure 5. 7: Inconsistency Occurring: Adding New Item to the EUC diagram....80

Figure 5. 8: Inconsistency Occurring: Change of Name to the Abstract Interaction....81

Figure 5. 9: Inconsistency Occurring: Change of Name to the EUC component....82

Figure 5. 10: MaramaAI Architecture....83

Figure 5. 11: Example of Trace interaction....84

Figure 5. 12: Example of Trace Back interaction from Abstract Interaction....85

Figure 5. 13: Example of Trace Back interaction from EUC component....86

Figure 5. 14: Example of Map To EUC interaction from Abstract Interaction....87

Figure 5. 15: Example of Index Checker interaction of Abstract Interaction....88

Figure 5. 16: Example of Index Checker interaction of EUC component....88

Figure 5. 17: The scenario “Voter Registration” [6] used for the Evaluation....90

Figure 6. 1: Outline of our Requirement Quality Management Process....98

Figure 6. 2: Example of extracting an EUC model then adding a new abstract interaction....102

Figure 6. 3: Change the ordering of EUC elements 104

Figure 6. 4: Deletion of an EUC element 105

Figure 6. 5: Example of EUC interaction pattern template (1) and Visual differencing (2) 107

Figure 6. 6: Change generated EUC model following the EUC interaction pattern template 108

Figure 6. 7: MaramaAI tool architecture for managing consistency of requirement 109

Figure 6. 8: Example of Consistency Management: Delete Abstract Interaction 112

Figure 6. 9: Example of Consistency Management: Delete EUC Component 113

Figure 6. 10: Example of Check Consistency with a Template for the Generated EUC Model 114

Figure 6. 11: Example of Check Keyword of abstract interaction in the abstract interaction Component 115

Figure 6. 12: Example of Check Keyword of abstract interaction in the EUC Component 116

Figure 7. 1: Example of EUI prototype iterates from Essential Use Cases (EUC model (Ambler [10, 11]) 120

Figure 7. 2: Example of Scenario “getting cash” and its EUC diagram 121

Figure 7. 3: End-to-end EUC and EUI prototyping approach 125

Figure 7. 4: An example of performing mapping of EUC model to EUI prototype using the UI Pattern library with trace-forward/ trace-back and translating the EUI prototype to the concrete UI-HTML form 126

Figure 7. 5: Trace forward and Trace-back from EUC model to EUI prototype. 131

Figure 7. 6: Marama EUI and concrete UI view in a form of form- based UI 132

Figure 7. 7: Modification of EUI prototype - Addition and Deletion in EUI prototype 133

Figure 7. 8: MaramaEUI tool architecture. 134

Figure 7. 9: Example of Map EUC to EUI 136

Figure 7. 10: Example Trace Back from EUI prototype to EUC Model 137

Figure 7. 11: Example of Generating HTML form from the EUI prototype 138

Figure 7. 12.: User study results of Marama EUI-Usefulness, ease of use, ease of learning, satisfaction and accuracy 140

Figure 8. 1: Example of User Scenario: Reserve a Vehicle [1] 143

Figure 8. 2: Capturing requirements - trace the abstract interaction, trace back and map to EUC model 144

Figure 8. 3: Add New Item to Abstract Interaction 145

Figure 8. 4: Change of Abstract Interaction Sequence Ordering 146

Figure 8. 5: Delete the EUC component 147

Figure 8. 6: Visual differencing to check for incorrectness and incompleteness 148

Figure 8. 7: The generated EUI prototype (1) and translated HTML form (2) 149

Figure 8. 8: Modifications in Prototypes 149

Figure 8. 9: Trace back which performs from the EUI prototype 150

Figure 8. 10: Example of User Scenario in a Form of Use Case Description: Check-out books of a LLB system [2] 152

Figure 8. 11: Capturing requirements-trace the abstract interaction and map to EUC model 153

Figure 8. 12: Deletion of Abstract Interaction 155

Figure 8. 13: Change of EUC component Sequence Ordering 156

Figure 8. 14: Visual differencing to check for incorrectness and incompleteness 157

Figure 8. 15: EUI Prototype with the extension components (1) and the generated HTML form with hyperlinks (2) 158

Figure 8. 16: HTML main page and hyperlink pages generated from EUI prototype 159

Figure 8. 17: Example of User Scenarios in the form of Use Case Descriptions: Manage Venue and Manage Event review from the Silicon Dreams Event Listing System specification 161

Figure 8. 18: Capturing Requirements - Trace the Abstract Interaction, Trace Back and Map to EUC model with Multiple Requirements 162

Figure 8. 19: Change of Abstract Interaction Name 163

Figure 8. 20: Delete EUC component 164

Figure 8. 21: Visual differencing to check for incorrectness and incompleteness for first scenario 165

Figure 8. 22: Visual differencing to check for incorrectness and incompleteness for second scenario 166

Figure 8. 23: Multiple EUI prototypes and HTML forms 167

Figure 8. 24: Changes made to the EUI prototype (1) and the results in HTML form (2)............ 168

Figure 9. 1: Usability Results-Capturing Requirements 181

Figure 9. 2: Usability Results - Consistency Checking of Requirements 187

Figure 9. 3: Usability Results- End to End Rapid Prototyping.. 192

Figure 9. 4: Comparison Results of Usability Study for Capturing Requirements, Consistency Checking and End-to-End Rapid Prototyping of MaramaAI 196

Figure 9. 5: Comparison results of CD Study for Capturing Requirements, Consistency Checking and End-to- End Rapid Prototyping of MaramaAI... 197

List of Tables

Table 2. 1: Comparison and Classification of RE Tool features..24

Table 2. 2: Type of Requirement Quality and its Description..26

Table 3. 1: EUC Extraction Study Results 49

Table 3. 2: Example of Abstract Interaction and its Related Category..52

Table 3. 3: Example of EUC Model Generated..53

Table 3. 4: Example of EUI Pattern Library for EUI prototype 53

Table 4. 1: Example of Abstract Interactions and their Associated Essential Interaction and Their Related Domains 58

Table 4. 2: Example of Essential Interaction and its Associated Abstract Interaction stored in the Essential Interaction Pattern Library...60

Figure 4. 3: An example of performing an essential interaction extraction to a EUC model and supporting trace-forward/trace-back 61

Table 4. 3: User Perception Characteristics and Questions Evaluating Them 64

Table 4. 4: Participants Open Feedback 65

Table 4. 5: Comparison result of correctness between Manual extraction (previous chapter) and Automated Tracing Tool 67

Table 5. 1: User Perception Characteristics and Questions Evaluating Them 91

Table 5. 2: Tool Usefulness Result...92

Table 5. 3: Tool Ease of Use Results 92

Table 5. 4: Participants Open Feedback 93

Table 6. 1: Examples of EUC Interaction Patterns 97

Table 6. 2: Overview of Comparing the Generated EUC model with EUC Interaction Pattern Template..111

Table 7. 1: EUI prototype Modelling Study Results 122

Table 7. 2: Example of EUI pattern Category and its related EUI pattern and it's associated Abstract Interaction from the EUC model 128

Table 7. 3: Examples of EUI Pattern template with its associated EUI Pattern and associated Domains in the EUI Pattern template library 129

Table 7. 4: User Perception Characteristics and Questions Evaluating Them 139

Table 9. 1: CD Dimensions and Meaning by Blackwell [157] 172

Table 9. 2: CD Notations Used and Questions Evaluating Them 174

Table 9. 3: Manual Extraction of EUC Study Result 178

Table 9. 4: Proficiency level of Using the Marama tool and Experience with Any Other Tool 180

Table 9. 5: Evaluation Results for Cognitive Dimensions Questions 182

Table 9. 6: Open-Ended Feedback 185

Table 9. 7: Evaluation Result for Cognitive Dimensions Questions 188

Table 9. 8: Open-ended Feedback 190

Table 9. 9: Evaluation Result for Cognitive Dimensions Questions 193

Table 9. 10: Open-ended Feedback 195

Chapter 1: Introduction

1 Research Background

1.1 What is a Requirement?

A set of requirements is interpreted at the early phase of a system development [5] and it reflects the client's need for a system [72]. It describes "*how the system should behave, constraints on the system's application domain information, constraints on the system operation or specification of a system property or attribute*" [5]. Software requirement specifications elaborate the functional and non-functional requirements, design artifacts, business processes and other aspects of a software system. Software requirement specifications that are complete and accepted by developers and clients provide a shared understanding and agreement of what a software system should do and why. Since requirement documents form the basis of development processes and this agreement, they should be correct, complete, and unambiguous [13] and need to be analysed with respect to Consistency, Completeness and Correctness ("3 Cs") to detect errors such as inconsistency and incompleteness [18].

Most requirements in the software industry are widely written or described using natural language. According to Fabrini et al. [14] at least the first level of the system is described using natural language. Requirements described using informal natural language are commonly written in narratives or scenarios. The major disadvantages of specifying requirements only in natural language "*are inherent imprecision, such as ambiguity, incompleteness and inaccuracy*" [13]. It has also been found that they are often error-prone which is partially caused by interpretation problems due to the use of natural language itself [14]. Although the development of object-oriented analysis, e.g. using (semi-)formalised models like UML [15] or formal models like KAOS [16], has afforded better requirements specification [16,57], most requirements documentation or software system specifications are still often written in – or at least derived from - free text expressed in natural language. As a result, this leads to requirements that are vague, informal and contradictory and that may or may not express the users' needs. In addition, Zowghi et al. [17] argue that it is difficult, costly and time consuming to maintain the consistency of the entire software requirements specification if that specification is derived from natural language.

1.1.1 Consistency

There are several definitions of consistency relating to software requirements specification. These definitions clarify what consistency is and when it appears in a software requirement specification. Zowghi and Gervasi state that consistency requires that no two (or more) requirements in a specification contradict each other, where there is no case that the requirements cannot be satisfied at the same time [18]. They also stress the importance of terminology, i.e. that words and terms always have the same meaning throughout the requirement specification. Both these views entail the need for ways of avoiding mutually exclusive statements and conflicts in terminology [13]. Liu [19] asserts that a specification is consistent when there is a computational model for its implementation and the specification will be valid when it ensures the user requirement. Consistency is also present when there is no internal (logical) negation between the specifications of a system [20]. A few types of consistency apply to specifications, including the precondition of a function being satisfied by the function calls, subtypes that include arguments of functions, and results of function subtypes [20]. Some relate to consistency between various non-functional requirements e.g. that security, reliability, scalability and platform requirements can all be met by the requirements as captured. In order to make sure requirements are always consistent and follow the customer's needs from the beginning, consistency checking needs to be done from the earliest stage of the Requirement Engineering process: Requirements Analysis (RA) as shown in Figure 1.1.

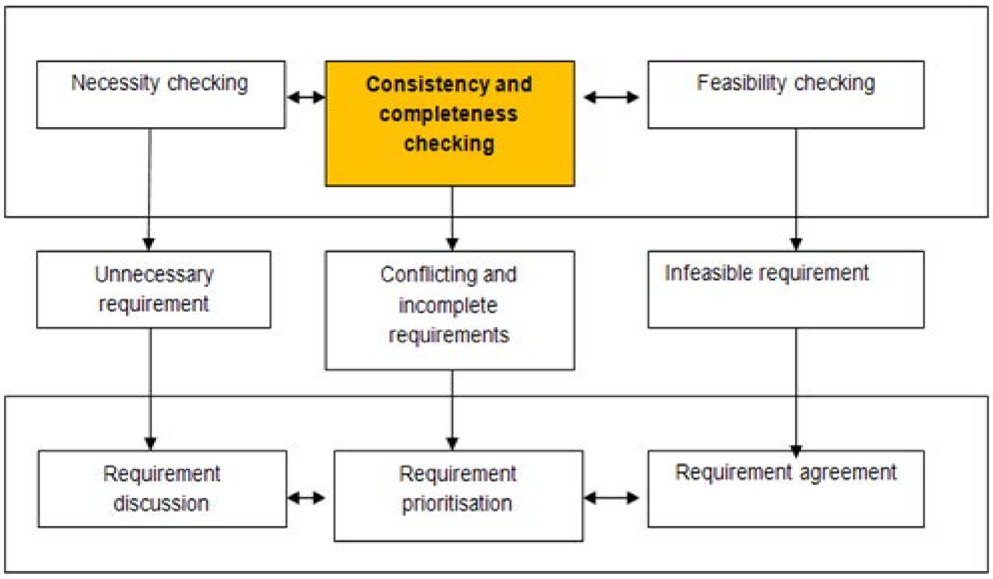


Figure 1.1: Requirement Analysis and Negotiation Process (From [5])

1.1.2 Inconsistency

It is common to find inconsistencies in requirements specifications as the requirement elicitation process involves two or more parties in delivering and understanding correct requirements. Zowghi et al. [18] assert that expression by different stakeholders may lead to inconsistencies and contradictions because the parties keep changing their minds throughout the development process. *Inconsistent requirements* occur when two or more stakeholders have differing, conflicting requirements and/or the captured requirements from stakeholders are internally inconsistent when two or more elements overlap and are not aligned [21], [22]. Typically the relationship is articulated as a consistency rule against which a description can be checked. Inconsistency in requirements also occurs when there are incorrect actions [14], or where requirements clash because of disagreements about opinions and bad dependencies [20], sometimes resulting from a lack of skills or the capabilities of different users dealing with shared or related objects. In addition, Litvak [23] believes that inconsistency occurs when the same parts of the model are portrayed by multiple diagrams and Lamsweerde et al. [24] find that inconsistency occurs in a set of descriptions when the descriptions can't be satisfied all together.

Overall, in our context of work, inconsistencies happen when any of the requirements components that are intended to be equivalent are not; this could be by not being in the same sequence, not having the same name, not being consistent when equivalent components are changed and not being consistency across differing representational models.

Positive and negative outcomes for the system development lifecycle are caused by having inconsistency [21]. The inconsistency helps to highlight the contradictory views, perceptions and goals among stakeholders who are involved in a particular development process. It also helps to identify which part of the system needs further analysis, as well as helping to facilitate the discovery and evocation of the options and information of a system. In addition, Nuseibah et al. believe that inconsistency can be used as an assisting tool to verify and validate the software process [22].

However, it is still vital to avoid or check for inconsistency as it could affect the whole development process, as the clients' requirement needs by the client cannot be met and attempts to do so may cause delay, increase the cost of the system development process's costs, put at risk the properties related to the quality of a system and make the maintenance process of a system cumbersome.

1.2 Research Motivations

Requirements captured in natural language are normally vague and error-prone due to the interpretation problem [14]. This is because the capturing process involves a human-centric representation which is full of arguments and misunderstandings [11]. The process of collecting the information for specific requirements may also take a long time as the requirements need to be gathered until they satisfy the client, and this process needs to match the client's available time [25]. There are also circumstances where requirements analysis ends prematurely because of delays and impatient clients [25]. This encourages validation to be effected at an earlier stage of requirement analysis in order to make sure the captured requirements are valid. Besides, waiting for late validation may cause the requirements' quality to suffer [25]. This problem leads to requirements' quality problems such as inconsistency, incorrectness and incompleteness.

However, as stated by Zowghi and Gervasi, "*improving the consistency of the requirements can reduce the completeness and, thereby again diminish correctness"* [18]. Therefore, consistency is of great importance to ensure the requirements are entirely precise and fulfil the needs of a user. In order to check and maintain consistency and diminish inconsistency, many techniques have been used. These include traceability, formal analysis and semi-formal analysis [26],[27],[28],[29],[30]. Traceability is sometimes not applicable in practice as it is too difficult and costly [31], although it helps in a number of activities in software development such as the evolution of software systems, compliance verification of the code, requirements validation, aspect identification, and any design decision [32]. Further, engineers may not be able to foresee or visualise the results although automated traceability tools are provided [33],[119].

In addition, in many projects consistency and completeness checking is normally performed manually by a "*tedious procedure of reading the requirements documents and looking for linguistic errors*" [34]. Many of these approaches to requirements consistency checking require heavyweight formal approaches where requirements must be expressed in complex formal models. While these are important in many domains e.g. safety-critical systems, they have proved difficult to put into widespread use [35]. Similarly, traditional approaches to using natural language processing and analysis of textually expressed requirements requiring the use of complex analysis algorithms and the complexity of natural language and its inherent ability to express inconsistent statements makes this challenging [36].

Translating requirements into semi-formal models, e.g. UML use cases, is a common approach that supports some limited analysis while improving the structure of the natural language expressed by requirements. However, translating these semi-formal models and checking consistency between them and natural language requirements have continued to prove problematic [37]. Besides, these works are difficult enough for requirements engineers to understand let alone clients or stakeholders, who are mostly non-technical or non-IT people: most clients do not understand models, formal terms or mathematics equations [38]. Furthermore, some natural language is interpreted differently from its original intention by requirements engineers [38].

As determined by various studies, eliciting requirements and extracting their use cases can be arduous and can lead to a rather imprecise analysis [39],[40],[41],[42]. Constantine and Lockwood [4] were thus motivated to develop the Essential Use Case (EUC) modelling approach to overcome some of these problems. Although the usage of EUCs is not as widespread as conventional use cases, several researchers have recommended their adoption as they helps to integrate the requirements engineering and interaction design processes [39],[43],[44]. Some of the main reasons EUCs are not commonly used are: a lack of tool support, engineers' lack of experience in extracting essential interactions from requirements, and a lack of integration with other modelling approaches [39], [43]. A further study to confirm the problem with more qualitative results is discussed in Chapter 3.

We have been motivated by the work done by Constantine and Lockwood [4] in developing the Essential Use Cases (EUC) modelling approach, as EUCs are beneficial in integrating the requirements engineering and interaction design processes to mitigate consistency issues between the requirements and design artefacts and to improve the traceability [43]. We have attempted an approach which applies traceability to the EUC concept, in the process managing the consistency and supports validation of requirements.

Overall, the motivation of this research is to provide automated support and a more lightweight approach for capturing requirements written in natural language and to manage the consistency and validation of requirements for various domains and applications with less human intervention and complexity. The aim is to provide notification and visual support for detecting inconsistency, as well as to determine other requirements quality errors such as incompleteness and incorrectness. In addition, the focus is to provide end-to-end support for both the requirements engineer and the client in confirming the consistency of requirements.

1.3 Research Questions

The main research question in this research in relation to our research motivation is:

> "***Can automated support enable us to better manage the consistency and validation of requirements?"***

In order to address this major research question, we have divided it into smaller research questions as follows.

1. *"Can a lightweight extraction process with automated tool support extract quickly and accurately essential requirements (abstract interactions) from textual natural language requirements?"*

   This question focuses on how to appropriately handle natural language requirements. To answer this question, we propose a lightweight extraction approach and its implementation in an automated tracing tool. To evaluate the approach we examine tool performance and efficacy in handling the extraction and user perceptions regarding the tool's usefulness and ease of use. This is addressed in Chapter 4 of this thesis.

2. *"Can the automated tool support the consistency and validation of requirements?"*

   This question focuses on managing consistency and validating requirements quality – consistency, completeness and correctness. To answer this question, we propose and implement automated traceability and visualization support to check the consistency of requirements in three different forms; textual natural language, abstract interaction and Essential Use Case models as well as to further validate the correctness and completeness of requirements. In addition, we evaluate the user perception of the tool's usability and the perceived strengths in selected dimensions of Cognitive Dimensions (CD). These aspects are addressed in Chapters 5, 6, 8 and 9 of this thesis.

3. *"Can the generation of UI prototypes from EUC models support the end to end validation of requirements between the requirements engineer and client?"*

   This question focuses on extending the tool to provide end-to-end support of requirements consistency checking, usable by both the requirement engineers and clients to confirm the consistency of requirements. To answer this question, we propose and implement a rapid prototyping approach embedded in our tool which is able to generate EUI prototype models and concrete UI HTML views from EUC models. In addition, we evaluate the tool usability and user perceived strengths and apply the tool to several case studies in different domains. These aspects are addressed in Chapter 7, 8 and 9 of this thesis.

In brief, in order to answer these research questions, we adopt a lightweight automated approach, and its realisation as a tool, to support consistency management and validation of requirements. We evaluate the tool efficacy, mainly in the extraction process, and the user's perception of the tool and its application.

1.4 Research Objective

The main objective of our research is to provide a requirement management approach that supports the consistency and validation of requirements from the early stages of Requirement Analysis. In particular, the research aims to provide the following.

1) To better support users and developers to work with informal and semi-formal requirements and keep them consistent. This will assist requirement engineers or business analysts to check whether their requirements, written or collected in natural language, are consistent with other analysis and design representations.
2) To support Requirements Analysis in order to improve requirements consistency and quality. This will use a set of essential interactions and EUC interaction patterns together with visual differencing to assist engineers in finding appropriate abstract interactions to design the EUCs for a particular system.
3) To provide end-to-end support to confirm the requirements consistency from both the requirement engineer and client's perspectives. This will use end-to-end rapid prototyping to visualise the requirements captured by a requirement engineer in the form of an abstract EUI prototype and concrete UI view in the form of an HTML page.
4) To provide traceability of requirements from both informal, semi-formal requirements and a UI prototype. This will assist a requirement engineer to trace forward/ trace–back from the different requirements representations to make sure the requirement is consistent.
5) To provide a proof of concept tool to allow automation with visualisation support in managing requirements consistency and validation of requirements in various domains of application. This will lessen human intervention in managing and validating the requirement.
6) To assess the consistency management approach by performing tool efficacy and end-user evaluations.

1.5 Research Methodology

As it is commonly used in software tool research, we have adopted an iterative approach, using successive iterations of tool development and evaluation to address our research questions [120],[164]. This can be categorized as an adaptation of action research [165],[166]. The focus of this latter methodology fits well with our work as it comprises a cycle of changes, evaluations and reflections [166]. We follow the cycle in our research research by realising our research knowledge with tool development and then evaluate the tool to gain feedback from the end-user experience. The five components of this research are:

i. Diagnosing: Here, we diagnose the problems faced from our preliminary study and literature, or from preceding cycle results

ii. Action planning: Here, we plan the problem-solving for the identified problems

iii. Action taken: Here, we try to resolve the problems by performing a solution. In our case, we develop the tool iterations and supporting assets, such as interaction libraries.

iv. Evaluating: We then evaluate the solution via end-user studies.

v. Specifying learning: From here, we identify the strength and weaknesses

In order to perform our work we employ additional extraction, consistency management, traceability and analysis components after evaluation of each stage of the research. An outline of our key steps follows.

- We have conducted a literature review of consistency and inconsistency checking of requirements in the Requirement Engineering domain, comparing and evaluating their approaches for checking inconsistency in requirements.
- From this, we identified an initial concept, outlined above, of how to support the checking of requirements inconsistencies providing traceability and aspects of completeness and correctness checking. The initial functional requirements are elaborated using scenarios or use case descriptions, which were collected from published material such as software engineering and requirements engineering books, proceedings and journals and published software developers' page.
- We then collected and categorised the natural language terminology following the interaction patterns of Essential Use Case from different case studies and scenarios and produced a database of key abstract interactions.
- We developed an initial automated prototype to explore the problems and issues which extracts and trace between textual natural language requirements and add to EUCs by using our database of abstract interactions.
- We also developed a set of consistency rules between the textual natural language requirements and the Essential Use Case model of requirements.
- We have identified appropriate usage scenarios and evaluated the result of using our consistency management and tracing tool if changes are made to the requirements.
- We developed an initial prototype of our automated inconsistency checking tool by embedding the tracing tool in Marama and connecting it to Marama Essential and Marama EUI, the User Interface design tool.
- We evaluated the automated inconsistency checking tool by using case studies to derive scenario examples and conducted a preliminary end user study on the usability of the tool.
- We refined our prototype by adding further analysis support for requirements quality checking using the EUC interaction pattern; adding further inconsistency management and traceability support features; and eventually adding traceability and consistency management support to more requirements and UI models.

- We then evaluated the refined tool with a larger formal end user study assessing the usability of the tool.
- Finally, we derived conclusions from our review, refinement and evaluations.

1.6 Research Contributions

The main contributions from this research are as follows.

1) This research provides better support for requirement engineers and developers to work with informal and semi-formal requirements and keep them consistent. We produce a lightweight approach to deal with natural language requirement together with a proof of concept tool called an automated tracing tool, which provides authoring facilities for textual natural language requirements and checking the consistency of these requirements. We also produce an essential interaction library and a collection of abstract interaction and essential interaction patterns which are reusable and can be applied to various domains of application. This library helps to enhance the accuracy level of natural language requirement and to assist the generation of the EUC model. Papers describing this work are:

    - *"Automated Software Tool Support for Checking the Inconsistency of Requirements*" which was published in Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering.(ASE 2009), and
    - *"Tool Support for Essential Use Cases to Better Capture Software Requirements"* which was published in Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering. (ASE 2010).

2) This research provides requirements analysis support in order to validate the requirements' consistency and quality. We provide a set of essential interactions and EUC interaction patterns together with a visual differencing approach to assist engineers in finding appropriate interactions for designing the EUC for a particular system. We provide consistency management using a traceability approach for any form of requirement; a textual natural language requirement written in a form of user scenario, abstract interaction and EUC model. The checking process is assisted by a visual approach and a warning notification to highlight the existence of inconsistency and other requirement quality errors such as incompleteness and incorrectness. Papers describing this work are:

- *“MaramaAI: Tool support for capturing requirement and checking the inconsistency”* which was published in Proceedings of the 21st Australian Software Engineering Conference (ASWEC 2010);
- *“Managing consistency between textual requirements, abstract interactions and Essential Use Cases*” which was published in Proceedings of the 34th Annual IEEE International Computer Software & Applications.*(COMPSAC* 2010);
- *“Marama AI: Automated and Visual Approach for Inconsistency Checking of Requirements”* which was published in Proceedings of the 18th Requirement Engineering Conference. (RE 2010), and
- *“Improving Requirement Quality using Essential Use Case Interaction Patterns*” which was published in Proceedings of the 33rd International Conference on Software Engineering. (ICSE 2011).

3) This research provides end-to-end rapid prototyping support to help confirm that the requirements which have been captured by a requirements engineer are fully consistent with the client’s original requirement. We provide an approach to map automatically the semi-formal requirements in the form of an EUC to abstract an Essential User interface (EUI) prototype model, and a more concrete UI view in the form of a HTML page. Traceability support is also provided to allow trace forward and trace-back between the EUC model, textual natural language and EUI prototype. A set of EUI patterns has been developed for enhancing the accuracy of the generated EUI prototype.
    - *“Generating Essential User Interface to Validate Requirements”* which was published in Proceedings of the 26th IEEE/ACM International Conference on Automated Software Engineering. (ASE 2011).
4) This research developed a prototype of an automated inconsistency checker called MaramaAI which was embedded in the existing Marama meta-tool; and which acts as a proof-of-concept of our approach. We evaluated the prototype using an end-user study confirming the usability of the tool based on the Cognitive Dimensions (CD) approach [158] . A conference paper was presented describing this approach:
    - *“Improving Requirement Quality using Essential Use Case Interaction Patterns*” published in Proceedings of the 33rd International Conference on Software Engineering. 2011 (ICSE 2011).

1.7 Thesis Organisation

The following chapters are organised as follows.

Chapter 2: Related Research
This chapter discusses key related research on the background of Requirement Engineering, the Requirement Engineering Tool, Requirement Validation, and Consistency Management which concentrates on the techniques used to check the requirement consistency and inconsistency. The techniques are compared and analysed and the limitations and gaps for each technique are identified.

Chapter 3: Motivation and Overview of Our Approach
This chapter presents an in-depth analysis of the problems that motivated our research and describes an overview of our approach to end-to-end support to address the identified problems.

Chapter 4: Essential Interaction Extraction
This chapter presents our lightweight approach for capturing the essential requirements from textual Natural language requirement using the essential interaction extraction and essential interaction library.

Chapter 5: Managing Requirement Consistency
This chapter describes our approach for managing the consistency among three forms of requirements; textual natural language, abstract interaction and the Essential Use Cases (EUC) model.

Chapter 6: Improving Requirement Quality using the Essential Use Case Interaction Pattern
This chapter describes our approach to further checking consistency and other requirement qualities such as completeness and correctness using the EUC Interaction Pattern and visual differencing.

Chapter 7: End-to-End Rapid Prototyping
This chapter describes our end-to-end rapid prototyping approach to help confirm that the requirements captured by a requirement engineer are consistent with the client's original requirements.

Chapter 8: Case Studies Examples
This chapter describes three different case studies of requirements written in a form of user scenarios that we use to demonstrate and describe the key features of our proof concept tool: MaramaAI.

Chapter 9: Evaluation
This chapter discusses the evaluation results gained from a formal evaluation conducted with end users.

Chapter 10: Conclusion and Future Works
This chapter summarises our research achievements and proposes future research directions.

1.8 Summary

This chapter discusses the core research, the implementation of automated support in managing the consistency and validating the requirements. We have presented our research motivation and the methodology adopted. We also described the contribution of the research in general and the composition of later chapters.

Chapter 2 Related Research

This chapter discusses requirement management and modeling techniques in general and describes the overview of the requirement specifications and semantics used in a requirement. An overview of requirements engineering tools (RE tools), sometimes called requirements management tools, is also presented. The key features of RE tools are classified and compared. This leads to a discussion of the requirements validation process and further key related research on consistency management of requirements. Work related to consistency is then analysed and discussed. The chapter concludes with an outline of an appropriate programme of work for checking inconsistencies, developed from the outcomes of the analysis and discussion.

2.1 Requirement Management

Requirements are often unstable [45] as many defects occur, such as conflicts, inconsistencies and incompleteness [46]. A document-based requirement specification approach also constrains the flow of requirements as it is complicated to keep up-to-date and it is always difficult to inform the stakeholders of any changes made [45]. Storage of newly added information and links between requirements and design such as use cases are also identified as cumbersome [45]. To overcome these problems, requirements management is essential. This involves activities such as finding, organising, documenting and tracking the requirements for a software system [47]. Requirements management is vital from the beginning of system/software development as it responds to changes and deals with the result of the changes [48]. Managing requirements is not limited to managing change but also manages the multiple configuration of requirements, requirement versions and requirement deliveries based on the allocated time, cost and correct quality [49]. Furthermore, requirements management relates to documentation and ensures that changes are made consistently across sets of documents [47].

There are two types of requirements management, a narrow sense and a broad sense [48]. The broad sense focuses on managing requirements throughout the software life cycle, either during the development phase or after deployment. The narrow sense focuses on the management of changes in the requirements engineering domains [48]. Requirements therefore need to be managed in order to ensure their consistency, integration and correctness [48].

2.2 Requirement Modelling Technique

As discussed in Chapter 1, requirements are crucial before any system/software development starts. Thus, knowledge and reasoning from the earlier phase of requirements engineering is necessary [50]. Requirements modeling such as goal-oriented, aspect-driven and system requirements modeling can be used to capture the requirements [50, 51]. As also discussed in chapter 1, most requirements are written in informal natural language. Formal language is also used to illustrate the requirements to ensure the quality. For example, the KAOS language, which focuses on goal-driven modelling and methodology [16, 52] is able to capture not only what but also why, who and when in a requirement [16]. It also offers a rich ontology that can be used to specifically capture the "*goal, constraints, object, action and agent*" [16] of a requirement. The other common formal language used as a requirement specification is B specification [53]. It is particularly used to check the inconsistency of a requirement [53]. Most requirements techniques as well as the automated verification tools currently available are used for verifying and checking the requirement quality, such as completeness and consistency [50].

2.3 Requirements Specification

Most of the research on requirements documentation focuses on specification languages and notations, with a range of formal, semi-formal and informal language [54]. Requirements specification, which is commonly generated once the requirement analyst has consulted the user [13], can be represented either in a formal, semi formal or informal format, based on the purpose of the specifications.

2.3.1 Formal Specification

A formal specification is defined as *"the expression, in some formal language and at some level of abstraction, of a collection of properties some system should satisfy"* [55]. This definition covers the different notions reliant on coverage of the system, the types of properties, area of interest, the level of abstraction to be considered and the type of formal language being used [55]. It is also identified that a specification will become formal once it is expressed in a language consisting of three components: *"rules for determining the grammatical well-formedness of sentences (the syntax), rules for interpreting sentences in a precise, meaningful way within the domain considered (the semantics)* *and rules for inferring useful information from the specification (the proof theory)"* [55]. A formal specification is also closely related to the use of *"mathematics, logic or algebra"* [56] where the syntax, semantics and the manipulation of rules are clearly defined in the specification language [55]. Formal specification assists in reducing errors such as ambiguity and imprecision, which obviously leads to the inconsistency and incompleteness [16] of requirements. However, it remains challenging for average software engineers to use formal specifications to gain fast and visible outcomes [55]. In addition, formal specification is unable to stand alone and needs to integrate fully with other software products and processes throughout the software lifecycle [55].

2.3.2 Semi-formal Specification

A semi-formal Specification is a combination of diagram techniques and tabular techniques which represents information in a structured form as well as providing guidelines to structure the information by way of manipulation rules over the specification [56]. Here, modeling notations are commonly applied. Examples of models used are UML diagrams, such as the use case diagram, activity diagram, sequence diagram, class diagram, state diagram and collaboration diagram, and structured diagrams such as the data flow diagram, conceptual diagram, ER diagram and any other components and logic descriptions. Models are commonly used in managing the requirements because models can easily be decomposed into smaller parts, which allows them to be better understood [57].

2.3.3 Informal Specification

An informal specification is defined as the use of "*unrestricted natural language"* [56]. This type of specification describes and specifies system requirements by combining the use of graphics with semi formal textual grammar which is more "*English-like*" [58]. It acts as a vehicle to elicit user requirements and help the analyst and the client to communicate their understanding in verifying a particular requirement [58]. So far, it is the most commonly used method to represent requirements in industry [14]. However, as described in Chapter 1, it is often vague and error-prone [13, 14].

2.4 Semantics in Requirements

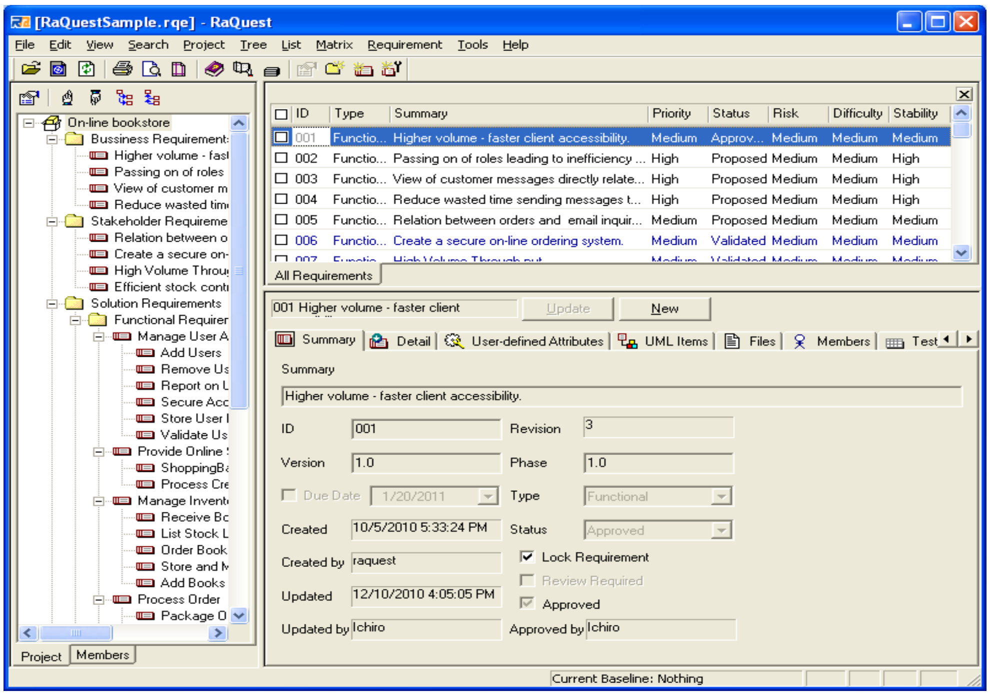
Apart from specifications, semantics are also applied to assist the requirements engineering process. Semantics is a study of words and sentences [59] which helps to handle a range of issues with the requirements analysis and indicates the classification, decomposition, terminology and prioritisation of requirements [59]. Further, semantics is also used as a term in a range of factors, any of which might provide the meaning of labels, functions and model decomposition [60]. Examples of common semantics are semantics of XML, of Natural language and of the web.

2.5 Requirement Engineering Tools (RE Tools)

In order to overcome the problems existing in the requirement engineering domain as described in Sections 2.1 and 2.2, especially in controlling and tracking the requirement changes, tool support is required [48]. Besides, software requirement specification (which involves different stakeholders and needs to fulfil many roles and interests) requires an automated tool to support collaborative requirement development processes [61]. Requirements engineering tools (RE Tools) are also commonly called requirements management tools [45]. There are currently three types of RE Tools in the market. First are tools which have existed for several years; the second are newly developed tools, and third are tools which are not designed for RE purposes but which are being used for RE activities such as Microsoft Office and Microsoft Excel [45]. This latter tool is becoming the most active market in the software development tool area [47]. There are two main categories of RE Tools: commercial and research tools. Available commercial tools support either the full requirements management process or just a part of the process [61]. Research tools tend to focus on a partial solution for a particular requirements management process.

2.5.1 Examples of Commercial Requirement Engineering Tools

There are many commercial requirement management tools available on the market: for example DOORS [62], Serena RTM [63], Caliber RM [64] and Requisite Pro [65]. These latter are leading the commercial market [45]. They provide rigorous coverage for requirements management but not for the requirements elicitation, analysis and validation [45] processes and so are not considered further here. Other tools include RaQuest [66], Q pack [67] and Enterprise Architect 8 [3]. We further evaluate these tools by self-exploration and the published information provided by the developers of these tools. RaQuest (see Figure 2.1) is a UML modelling tool which) manages the requirements of a system or an application, as well as tracking the changes in a requirement with various supported features [66]. It can also be used to generate documents for a whole project with different types of forms such as HTML, CSV, Word, Excel and RTF. For managing and defining a requirements item, the tool comes with a few facilities such as prioritisation, updating of log, definition of user attribute and assignation of members [66]. In order to allow requirements to be viewed at a glance, a project hierarchy and a list view are provided. For tracking the requirements, different types of relationships are applicable: for example, the relationship between the requirements item and *“connected”* requirements after any changes are made to the requirements; the relationships of requirements area displayed in a matrix view [66]. Overall, the tool provides rigorous requirements capturing facilities but does not provide any requirements validation facility. This is also supported by the survey reported by INCOSE [162].



- On-line bookstore
  - Bussiness Requirement:
    - Higher volume - fast
    - Passing on of roles
    - View of customer m
    - Reduce wasted tim
  - Stakeholder Requireme
    - Relation between o
    - Create a secure on-
    - High Volume Throu
    - Efficient stock conti
  - Solution Requirements
    - Functional Requirer
      - Manage User A
        - Add Users
        - Remove Us
        - Report on L
        - Secure Acc
        - Store User I
        - Validate Us
      - Provide Online !
        - ShoppingBa
        - Process Cre
      - Manage Invent
        - Receive Bo
        - List Stock L
        - Order Book
        - Store and M
        - Add Books
      - Process Order
        - Package O

☐	ID	Type	Summary	Priority	Status	Risk	Difficulty	Stability
☐	001	Functio...	Higher volume - faster client accessibility.	Medium	Approv...	Medium	Medium	Medium
☐	002	Functio...	Passing on of roles leading to inefficiency ...	High	Proposed	Medium	Medium	High
☐	003	Functio...	View of customer messages directly relate...	High	Proposed	Medium	Medium	High
☐	004	Functio...	Reduce wasted time sending messages t...	High	Proposed	Medium	Medium	High
☐	005	Functio...	Relation between orders and email inquir...	Medium	Proposed	Medium	Medium	Medium
☐	006	Functio...	Create a secure on-line ordering system.	Medium	Validated	Medium	Medium	Medium
☐	007	Functio...	High Volume Through put	Medium	Validated	Medium	Medium	Medium

001 Higher volume - faster client

Summary

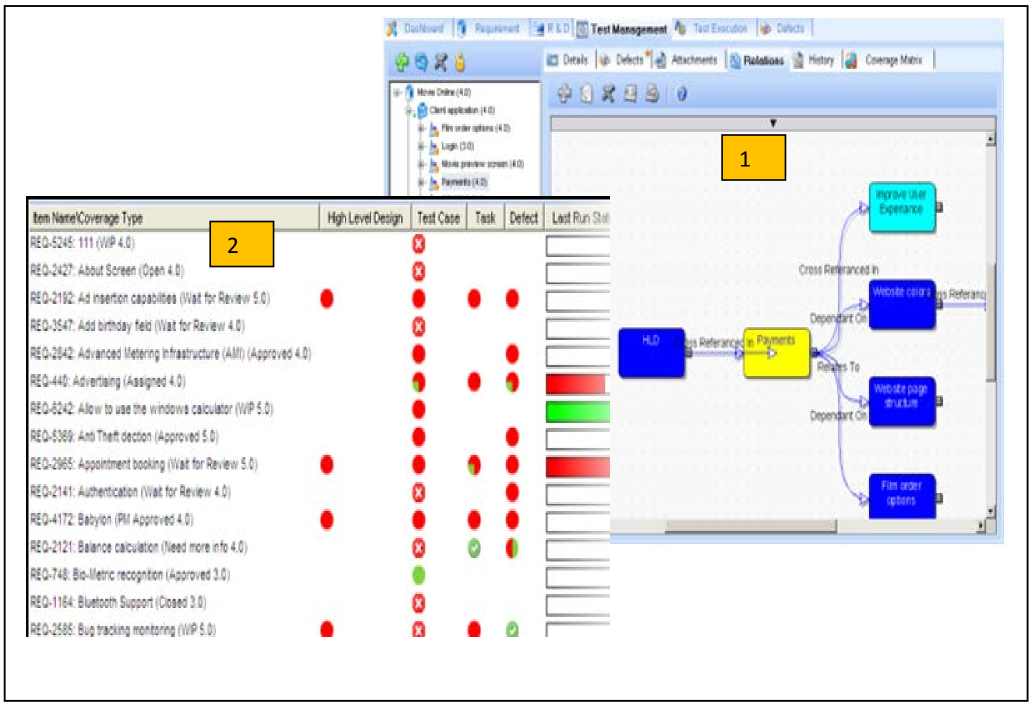
Higher volume - faster client accessibility.

ID	001	Revision	3
Version	1.0	Phase	1.0
☐ Due Date	1/20/2011	Type	Functional
Created	10/5/2010 5:33:24 PM	Status	Approved
Created by	raquest	☑ Lock Requirement	
		☐ Review Required	
Updated	12/10/2010 4:05:05 PM	☑ Approved	
Updated by	Ichiro	Approved by	Ichiro

Current Baseline: Nothing

Figure 2. 1: Example of Features in RaQuest [66]

Next is the QPack tool (see Figure 2.2) which provides better traceability, especially to gather business requirements, functional requirements or non–functional requirements, and also to track the changes made in the requirements using the testing coverage and defects produced by using QPack Analytic [67]. The QPack Requirements definition also acts as a single repository for requirements management purposes. For example, prioritising and estimating efforts, managing the complete life cycle and notifying changes in a requirement [67]. Furthermore, the solution is used to manage the software hierarchy and the traceability is measured using several KPIs [67]. The validation of a requirement is confirmed through testing and defect tracking. To document a requirement, the tool synchronises the changes made between the MSWord and the QPack requirements management repository [67]. Overall it is easy to use and is able to trace requirements well but provides limited requirements validation support by itself as it needs to be integrated with test management and defect tracking tools.



Item Name\Coverage Type	High Level Design	Test Case	Task	Defect	Last Run Stat
REQ-5245: 111 (WIP 4.0)		⊗			
REQ-2427: About Screen (Open 4.0)		⊗			
REQ-2192: Ad insertion capabilities (Wait for Review 5.0)	●	●	●	●	
REQ-3547: Add birthday field (Wait for Review 4.0)		⊗			
REQ-2842: Advanced Metering Infrastructure (AMI) (Approved 4.0)		●		●	
REQ-440: Advertising (Assigned 4.0)		●	●	●	🟥
REQ-6242: Allow to use the windows calculator (WIP 5.0)		●			🟩
REQ-5369: Anti Theft dection (Approved 5.0)		●		●	
REQ-2965: Appointment booking (Wait for Review 5.0)	●	●	●	●	🟥
REQ-2141: Authentication (Wait for Review 4.0)		⊗		●	
REQ-4172: Babylon (PM Approved 4.0)	●	●	●	●	
REQ-2121: Balance calculation (Need more info 4.0)		⊗	✓	●	
REQ-748: Bio-Metric recognition (Approved 3.0)		●			
REQ-1164: Bluetooth Support (Closed 3.0)		⊗			
REQ-2585: Bug tracking monitoring (WIP 5.0)	●	⊗	●	✓	

Figure 2. 2: Example of the Qpack Tool for (1) Requirement tracking and (2) Requirement traceability [67]

Enterprise Architect 8 (see Figure 2.3) is another example of a requirements management tool that assists users to capture requirements in detail and to manage changes that occur in a requirement [3]. It also provides a baseline to check for changes, deletions and additions which occur between processes, as well as version control to allow storage of the standard XMI text file of any compliant system [3]. Further, it provides links to different requirements assets such as to use cases, components, software artifacts, test cases and others [3]. Here, a complex traceability graph can be viewed for each requirement. In addition, it helps to produce detailed documentation and involves the whole team in defining or working on the captured requirements [3]. Overall we could sum up that this tool provides full derivation of requirements but only provides partial validation of them as it just identifies inconsistencies from the unlinked requirements using the provided traceability facility.

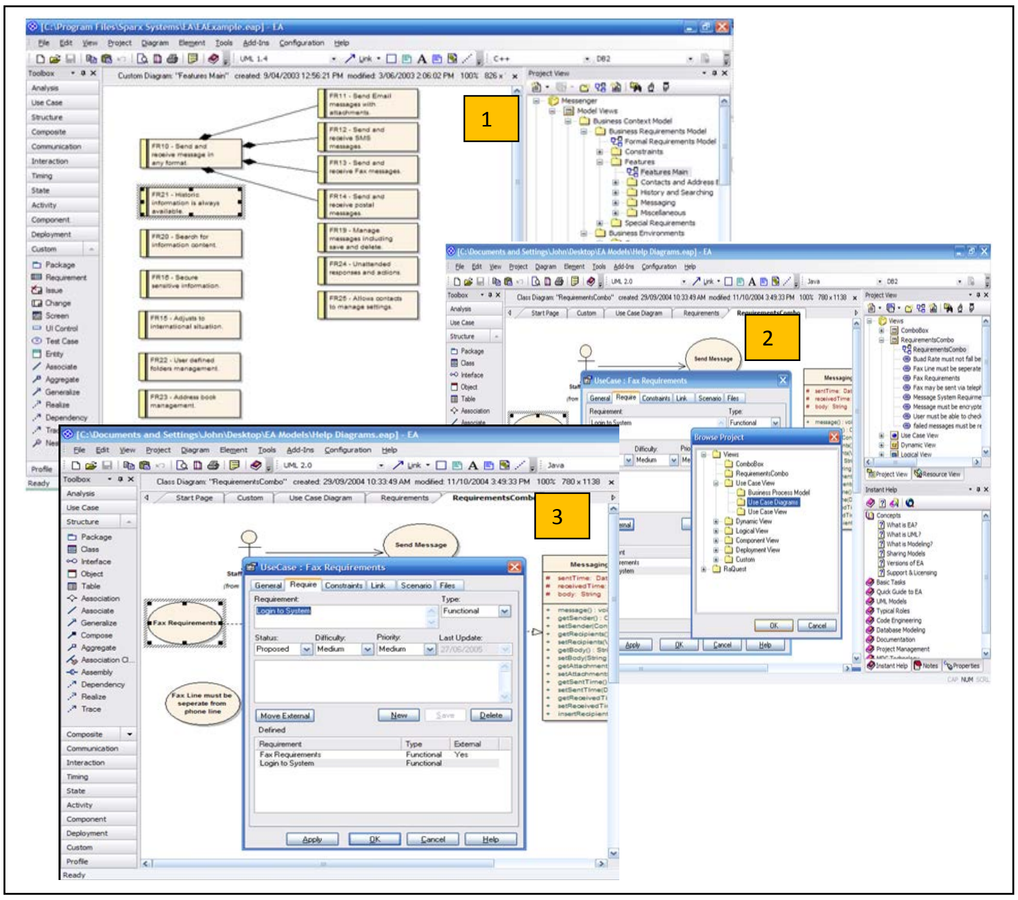
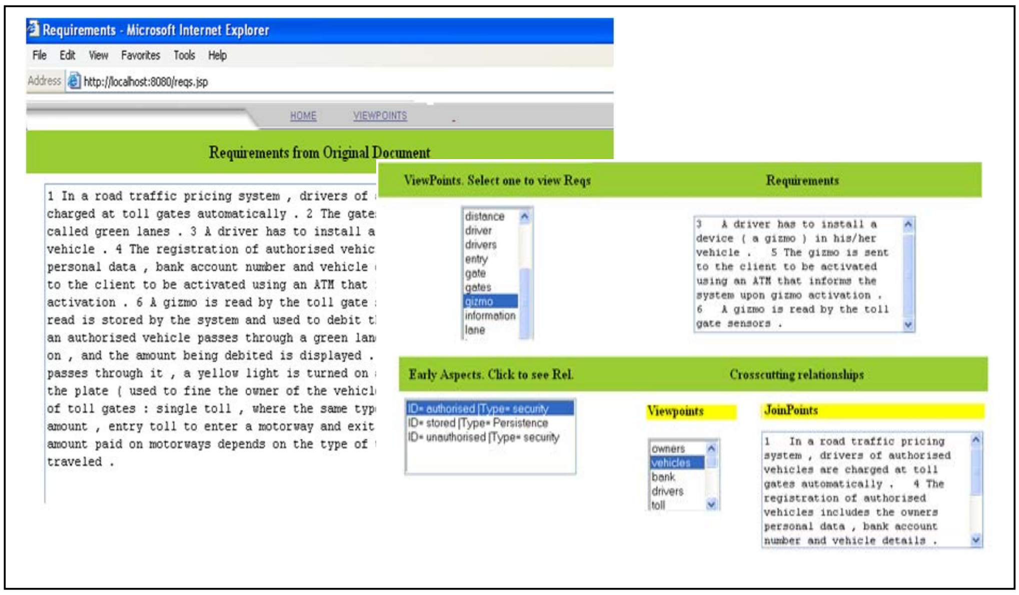

1
2
3
FR10 - Send and receive message in any format
FR11 - Send Email messages with attachments
FR12 - Send and receive SMS messages
FR13 - Send and receive Fax messages
FR14 - Send and receive postal messages
FR19 - Manage messages including save and delete
FR20 - Search for information content
FR21 - Historic information is always available
FR24 - Unattended responses and actions
FR25 - Allows contacts to manage settings
FR16 - Secure sensitive information
FR15 - Adjusts to international situation
FR22 - User defined folders management
FR23 - Address book management
Send Message
Fax Requirements
UseCase : Fax Requirements
Fax Line must be seperate from phone line

Figure 2. 3: Example of the Enterprise Architect 8 tool in managing: (1) requirements (2) internal requirements and (3) exporting the internal requirements [3]

Overall, commercial tools provide quite thorough support for managing requirements, especially in capturing requirements and managing changes that occur. But most do not provide a full checking or validation of requirements. Therefore, RE research tools have been developed to provide solutions for problems faced by the commercial tools.

2.5.2 Examples of Research Requirement Engineering Tools

As described in Section 2.5 research tools are more focused on partial solutions for particular requirements processes. For example, EA-Miner (see Figure 2.4), which is developed by Sampaio et al. [9], was mainly developed to identify and separate concerns, either aspectual or non-aspectual, with the relationships of their crosscutting at the level of requirements [9]. The tool supports four requirements processes. First is elicitation, where the tool assists by allowing the requirements engineer to focus on a particular section of the input documents and to help the RE rapidly gain understanding of the system [9]. Next is the identification of an activity where an internal representation is produced from the input file as a Java object for the use of specific techniques such as a viewpoint or scenario-based presentation of results or screening out of irrelevant abstractions [9]. The third process supported by this tool is presenting the results for the internal model in various ways, either in the form of diagrams or textual representation [9]. Finally, the tool also helps with the process of screening out and generating the requirement specifications documents by translating the model which has been refined to different formats such as XML, DOC and others[9]. However, this tool is in need of still further improvement to identify the early aspects of both functional and non-functional requirements, and also to enhance the functionalities of screening out to lessen the requirement engineer's efforts in using the tool [9].


Requirements - Microsoft Internet Explorer
File Edit View Favorites Tools Help
Address http://localhost:8080/reqs.jsp
HOME
VIEWPOINTS

Requirements from Original Document

```
1 In a road traffic pricing system , drivers of
charged at toll gates automatically . 2 The gate
called green lanes . 3 A driver has to install a
vehicle . 4 The registration of authorised vehic
personal data , bank account number and vehicle
to the client to be activated using an ATM that
activation . 6 A gizmo is read by the toll gate
read is stored by the system and used to debit t
an authorised vehicle passes through a green lan
on , and the amount being debited is displayed .
passes through it , a yellow light is turned on
the plate ( used to fine the owner of the vehicl
of toll gates : single toll , where the same typ
amount , entry toll to enter a motorway and exit
amount paid on motorways depends on the type of
traveled .
```

ViewPoints. Select one to view Reqs

Requirements

```
3   A driver has to install a
device ( a gizmo ) in his/her
vehicle .   5 The gizmo is sent
to the client to be activated
using an ATM that informs the
system upon gizmo activation .
6   A gizmo is read by the toll
gate sensors .
```

Early Aspects. Click to see Rel.

ID= authorised |Type= security
ID= stored |Type= Persistence
ID= unauthorised |Type= security

Crosscutting relationships

Viewpoints

JoinPoints

```
1   In a road traffic pricing
system , drivers of authorised
vehicles are charged at toll
gates automatically .   4 The
registration of authorised
vehicles includes the owners
personal data , bank account
number and vehicle details .
```

Figure 2. 4: Example of the EA-Miner tool in eliciting the input requirement(1) and identifying concerns (2) [9]

In addition, Adisa et al. [8] developed an open-access prototype tool called Living Requirement Space (LRS) to gather ERP system requirements using Web 2.0 technologies (see Figure 2.5). The tool helps to handle the problem of constantly changing business requirements characteristics [8]. It acts as a platform to allow collaboration at any stage of the requirements life cycle for all domain experts, business analysts and other ERP stake-holders [8]. The requirements life cycles involved are identification, analysis and management of business requirements, mainly for ERP systems [8]. It helps to collect, store, retrieve, control the versions and relationships with other requirements and to manage change requests [8]. The analysis of requirements is conducted via forums and discussion as well as prioritising requirements [8]. As it uses Web 2.0 technologies, it means that it is accessible to all users from various places, but there is still a problem if there is no internet connection or server available.

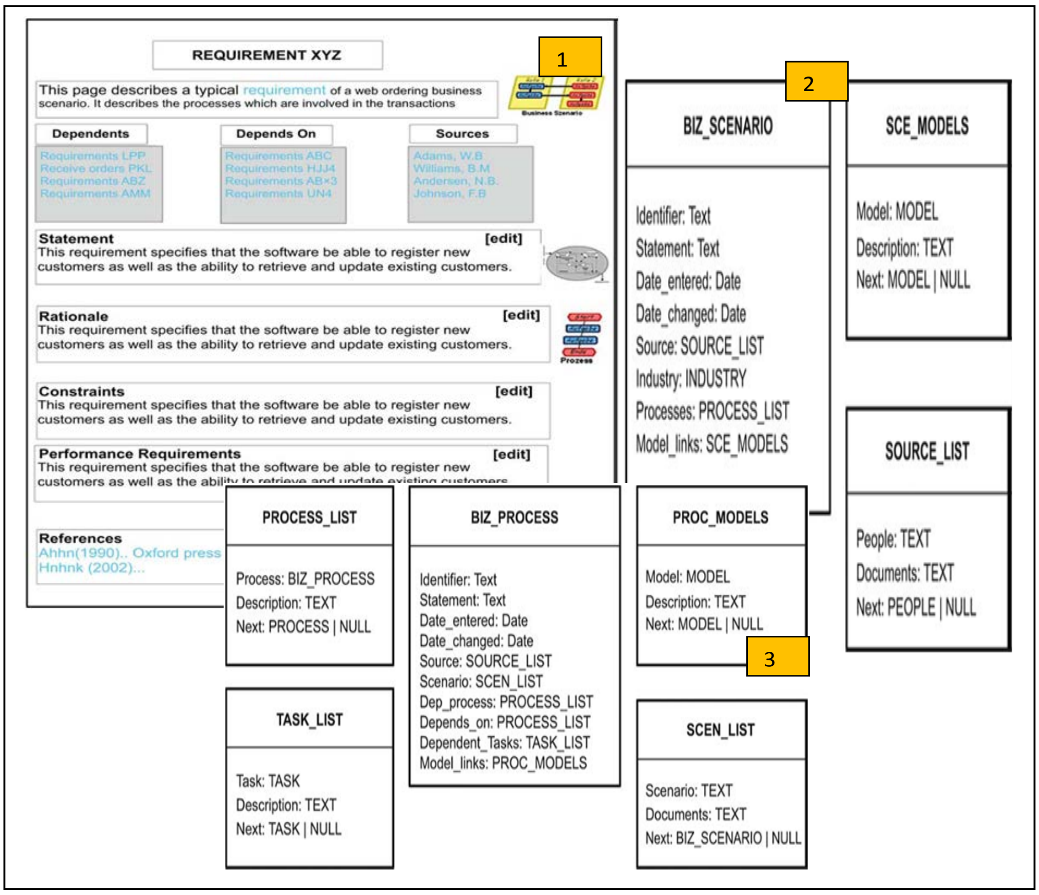


Figure 2. 5: Example of the LRS Requirement document (1), LRS scenario object and LRS task model (3) generated from the LRS tool [8]

Another example of a requirements management tool is the WikiReq system (see Figure 2.6), developed by Abeti et.al [12]. WikiReq is developed using wiki technology and is suitable to be used as a collaborative platform for discussion among stakeholders as well as for eliciting and managing requirements by using a semantic wiki [12]. The requirements are edited, argued and discussed between the stakeholders directly into the wiki. In order to have a rigorous elicitation of requirements, each requirement is acquired in a form of a Si* concept [12]. It also allows interoperability between semantic wiki and Integrated Development Environment (IDE) Eclipse to be achieved. The approach used in this tool also helps to reduce and simplify concepts which involve requirements and business processes as well as helping to maintain the coherence of requirements with the use of other technical artifacts, such as UML use cases and BPMN models [12]. The approach also relates the business requirements with the expected system starting with the use of use cases, business goals and business processes.

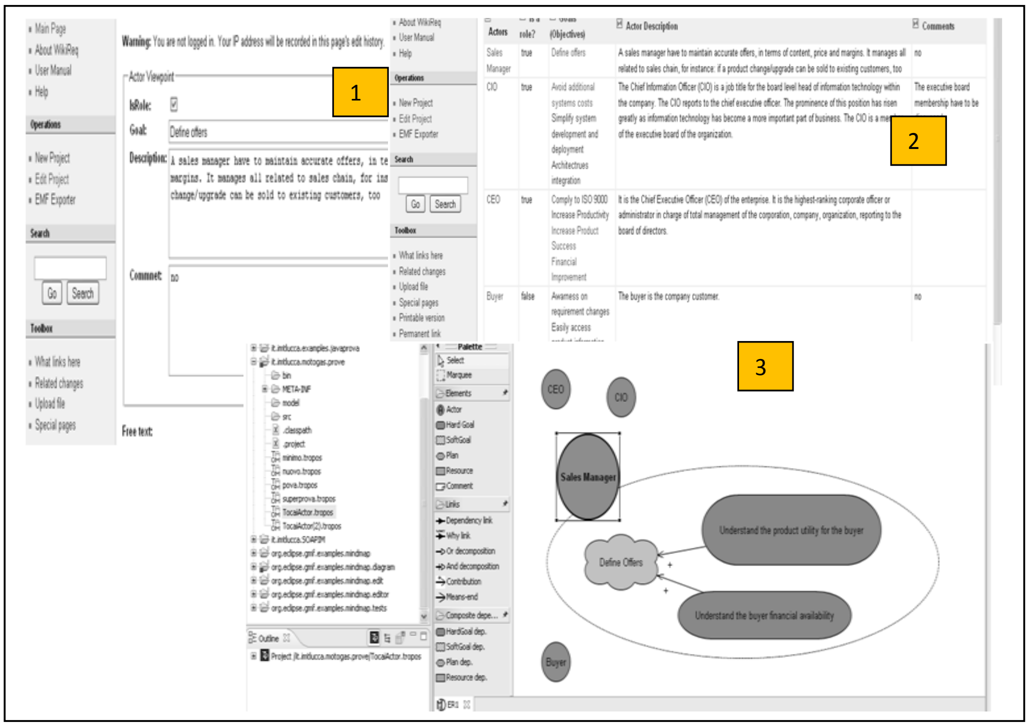
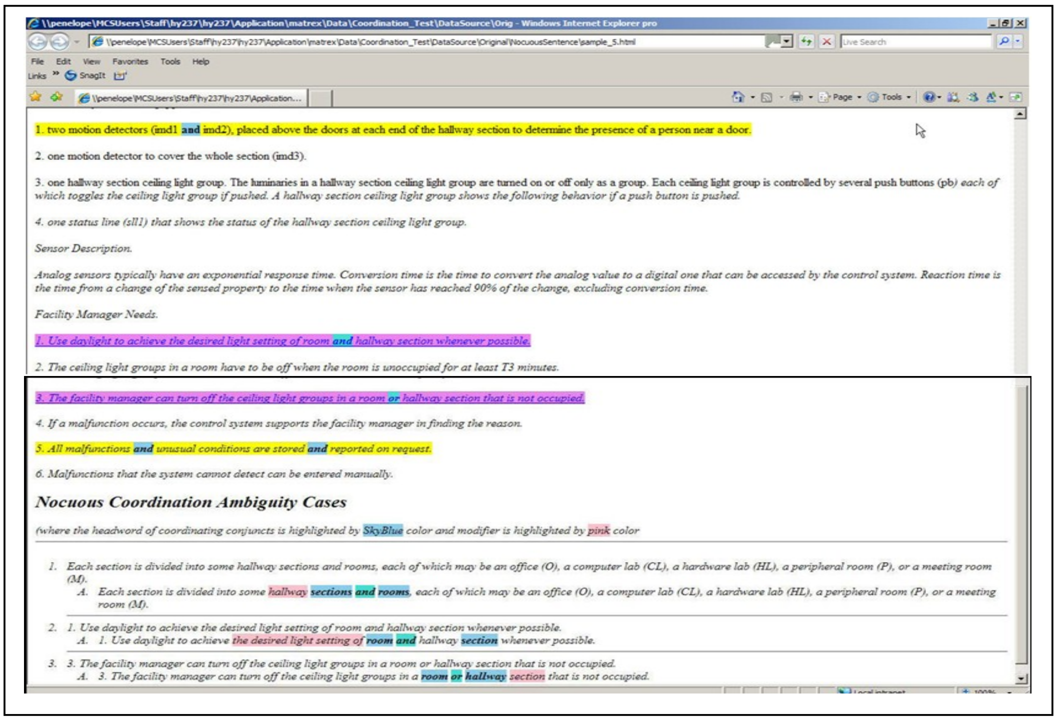


Figure 2. 6: Example of the WikiReq system in eliciting and managing requirement: (1) shows the actor view point page, (2) goal view point page and (3) the WikiReq exported to Eclipse [12]

Another automated requirements tool, developed by Yang et al. [7], is more focused than the other research tools described towards the requirement validation process. They developed a tool to automatically detect the "*nocuous*" coordination ambiguity in natural language requirements. Nocuous ambiguities are harmful ambiguities that lead to misunderstanding and to errors in an implementation [68]. They are recognized in any both conditions: present (acknowledged ambiguities) and undetected (unacknowledged ambiguities) [68]. The tool is called Nocuous Ambiguity Identification (NAI) (see Figure 2.7). This tool is able to identify ambiguity patterns as well as classifying the ambiguities; either as nocuous or innocuous cases by using the "*nocuity classifier*" [7]. The nocuity classifier employs a machine-learning algorithm called LogitBoost. If a nocuous ambiguity exists in the text, it is detected and highlighted on the screen by the tool as shown in Figure 2.7. The tool is effective and performs well based on the experimental results gained. However, the authors believe the tool's performance needs further improvement and the heuristics require further enhancement.

The focus of the tool also needs extension to a wider range of ambiguity types, not to be limited to only nocuous coordination ambiguity [7].


\\penelope\MCSUsers\Staff\hy237\hy237\Application\matrex\Data\Coordination_Test\DataSource\Orig - Windows Internet Explorer pro
\\penelope\MCSUsers\Staff\hy237\hy237\Application\matrex\Data\Coordination_Test\DataSource\Original\NocuousSentence\sample_5.html
Live Search
File Edit View Favorites Tools Help
Links SnagIt
Page Tools
Local intranet
100%

1. two motion detectors (imd1 **and** imd2), placed above the doors at each end of the hallway section to determine the presence of a person near a door.

2. one motion detector to cover the whole section (imd3).

3. one hallway section ceiling light group. The luminaries in a hallway section ceiling light group are turned on or off only as a group. Each ceiling light group is controlled by several push buttons (pb) *each of which toggles the ceiling light group if pushed. A hallway section ceiling light group shows the following behavior if a push button is pushed.*

4. one status line (sll1) that shows the status of the hallway section ceiling light group.

Sensor Description.

Analog sensors typically have an exponential response time. Conversion time is the time to convert the analog value to a digital one that can be accessed by the control system. Reaction time is the time from a change of the sensed property to the time when the sensor has reached 90% of the change, excluding conversion time.

Facility Manager Needs.

*1. Use daylight to achieve the desired light setting of room **and** hallway section whenever possible.*

2. The ceiling light groups in a room have to be off when the room is unoccupied for at least T3 minutes.

*3. The facility manager can turn off the ceiling light groups in a room **or** hallway section that is not occupied.*

4. If a malfunction occurs, the control system supports the facility manager in finding the reason.

*5. All malfunctions **and** unusual conditions are stored **and** reported on request.*

6. Malfunctions that the system cannot detect can be entered manually.

Nocuous Coordination Ambiguity Cases

(where the headword of coordinating conjuncts is highlighted by SkyBlue color and modifier is highlighted by pink color

1. *Each section is divided into some hallway sections and rooms, each of which may be an office (O), a computer lab (CL), a hardware lab (HL), a peripheral room (P), or a meeting room (M).*
   A. *Each section is divided into some hallway **sections** **and** **rooms**, each of which may be an office (O), a computer lab (CL), a hardware lab (HL), a peripheral room (P), or a meeting room (M).*
2. *1. Use daylight to achieve the desired light setting of room and hallway section whenever possible.*
   A. *1. Use daylight to achieve the desired light setting of **room** **and** hallway **section** whenever possible.*
3. *3. The facility manager can turn off the ceiling light groups in a room or hallway section that is not occupied.*
   A. *3. The facility manager can turn off the ceiling light groups in a **room** **or** **hallway** section that is not occupied.*

Figure 2. 7: Example of Results from the NAI tool in detecting nocuous coordination ambiguity [7]

2.5.3 Discussion of RE Tools Features

We have developed a characterisation of RE Tools, shown in Table 2.1, to illustrate their strengths and weaknesses. The classification criteria are based on our experience from exploring the tools and study of the published literature. The criteria were also informed by the INCOSE survey provided for Requirements Management tool [162] Classifications used are: processes (Elicitation and Analysis, Identification, Validation and Change Management), techniques used and type of specification used. These criteria are chosen because they are the common criteria that exist in most analyses of RE tools although they may be expressed in a variety of different phrases. This classification also leads us to clearly identify the gaps that still exist in the RE tools. This table shows that most RE tools discussed handle the elicitation and analysis processes as well as managing changes and identifying the requirements. But most of them do not handle the validation of requirements such as consistency, correctness and completeness, although Laplante [163] states that RE tools must include verification and validation processes and Yu [50] states that currently available requirements modeling techniques are mostly used for the validation process.Only NAI is very focused towards validating the ambiguity of requirements and QPack tries to detect the defects by using a testing mechanism. However, for these, it is proven that the validation process lacks tool support. Based on the classification too, most tools use semi-formal specifications, as an input and output to process and generate the results. The requirements techniques used also vary based on the process that the tool handles. We also conclude that most commercial tools are more interested in using or applying the traceability techniques, especially in tracking changes and eliciting requirements. The research tools use different techniques such as heuristics, early aspects, the wiki approach and interviewing and the brainstorming approach.

As there is little empirical research in validating requirements, we investigate further the key related research in validating requirements focused on the inconsistency problems. Firstly, however, the idea of requirements validation is discussed in general terms.

Requirement Tool	Processes				Technique Used	Type of Specification used	Source of Evaluation
	Elicitation & Analysis	Identification	Validation	Change management			
RaQuest	×	×	×	√	Features/ Relationship	Semi formal	Self-exploration/information from developer/INCOSE
QPack	√	×	√	√	Traceability	Semi formal	Self-exploration/information from developer
Enterprise	√	×	√ (partial)	√	Traceability	Semi formal	Self-exploration/information from developer/INCOSE
EA-Miner	√	√	×	×	Early aspects	Semi formal	Published literature
LRS	√	√	×	√	Interview, brainstorming,	Informal	Published literature
WikiReq	√	×	×	√	Wiki approach	Semi formal/ Formal	Published literature
NAI	×	×	√	×	heuristics	Informal	Published literature/self-exploration

Table 2. 1: Comparison and Classification of RE Tool features

2.6 Requirements Validation

Requirement validation is a process executed throughout the system life cycle [69]. It ensures the correctness, completeness and consistency of a requirement [69]. The descriptions of these, based on different points of view are shown in Table 2.2. The validation process also helps to determine that the end product is correct and complete as well as guaranteeing that the system developed satisfies the stakeholders' original requirements [69]. Late validation of requirements could cause requirement quality to suffer [25]. In order to make sure the original requirements of stakeholders are met, the requirements captured by the requirement engineer/analyst need to be entirely precise and consistent from the early stage of the RE process. Inconsistencies of requirements are identified as adverse and need to be avoided [22]. Hence, further related research to validate requirements focuses on managing the consistency/ inconsistency of requirements is discussed.

Type of Requirement Quality	Description
Correctness	"Describes the correspondence of that specification with the real needs of the intended users in much the same way that correctness of a piece of software refers to the agreement of the software part with its specification." [18] "A program is considered correct if it behaves as expected on each element of its input domain" [70]. "An SRS is correct if and only if every requirement represents something required of the system to be built" [168]
Completeness	"Implies that all customer's needs will be met when the system is constructed." [18] "A requirement must have all relevant components" [71] "A requirement's document should include requirements that define all functions and the constraints intended by the system user" [72] "It specifies required behaviour and output for all possible states under all possible constraints." [73] "Responses of the software to all realizable classes of input data in all realizable classes of situations is included" [167]
Consistency	"No two or more requirements in a specification contradict each other and the case where words and terms have the same meaning throughout the requirement's specifications (consistent use of terminology)" [18] "Requirement uses terms in a manner consistent with their specified meanings." [71] "Requirement should be understood precisely in the same way by every person who reads it." [71] "Requirements in the document should not conflict." [72] "Consistency is also referring to situations where there is no internal (logical) contradiction in a specification of a system." [20] "Consistent specification exists when there is a computational model for its implementation and the specification is valid when it satisfies the user requirements." [19] "An SRS is internally consistent if and only if no subset of individual requirements stated therein conflict" [167]

Table 2. 2: Type of Requirement Quality and its Description

Based on all the definitions, we sum up our understanding of consistency as it matches our work. Consistency happens when any of the requirements components are intended to be equivalent. The requirements components also should have the same naming and the sequence of the requirements need to be in the same order throughout the software requirements specification. In addition, consistency happens when the requirements captured by a requirements engineer are confirmed as satisfying the clients' intended need. We assume all the requirements are complete when there are no missing key definitions or constraints for the software system. We also assume all the requirements are correct when the requirements captured accurately, and with no redundancy, reflect the actual requirements and needs of clients.

2.7 Consistency/ Inconsistency Management

As discussed in Section 2.2, current available requirements techniques are used to verify and check the requirements qualities such as completeness and consistency [50], and to detect errors such as inconsistency and incompleteness. However, it was shown in Section 2.5.3 that few requirement tools available in the market provide facilities for the validation process. Thus, we would like to investigate related research done by others in validating requirements quality, especially consistency. Other researchers have devoted their studies to how to manage the consistency of requirements. There are efforts to check the existence of inconsistencies in either informal specifications, semi-formal specifications or formal specifications [17, 74], There is also work on managing the consistency in the architecture model and other design models [26, 51, 75]. In addition, there are heavyweight or lightweight approaches used to check for the inconsistencies. Further, there is also work on repairing inconsistencies and tolerating their presence [22, 76, 77]. There are many techniques to check for inconsistencies and to maintain the consistency of requirements [78]. Techniques used to check for consistency or to handle the inconsistency based errors include traceability and analysis approaches. Analysis approaches can be categorised as either formal analysis or heuristic analysis. Different types of specifications are also used to represent the requirements before consistency checking is conducted. Semantics is sometimes applied to the requirements to assist the validation process. We will explore work relating to these issues through the remainder of the sections.

2.7.1 Consistency/ Inconsistency Management Techniques in General

2.7.1.1 Traceability

Traceability is defined as the "*ability to describe and follow the life of an artifact which is developed during software lifecycle in both forward and backwards directions"* [79]. Traceability is an important approach to manage requirements effectively [5] and a vital practice in an organisation [31]. Traceability must also cover all aspects in terms of scope and coverage, including system level scope and all four types of coverage, as defined by Bashir et al [80]. First is the traceability between an origin of a requirement inclusive of source, stakeholders and requirements. Next, is traceability between the requirements and other requirements such as functional and non-functional requirements. Then, is traceability between the requirements and other artifacts which provides a trace between different requirements forms such as specifications, designs and test cases. Finally, is traceability between other artifacts and other artifacts such as considering links and dependencies among artifacts.

Cysneiros and Zisman (2008) assert that traceability relations help in a number of activities in software development [32]: for example, the evolution of software systems, compliance verification of code, requirements validation, aspect identification and any design decision. Traceability is often informally practised in tracing requirements to and from a software design [5]. Some traceability techniques are assisted by information retrieval (IR) a derived technique to support identifying traceability links. However, IR is unable to identify all links [32, 79]. Although traceability is important, it is sometimes not applied in practice as it is too difficult and costly [31].

2.7.1.2 Analysis Approach

There are two types of analysis identified for checking consistency/inconsistency: heuristic analysis and formal analysis.

2.7.1.2.1 Heuristic Analysis

"The term heuristic means a method which, on the basis of experience or judgement, seems likely to yield a reasonable solution to a problem, but which cannot be guaranteed to produce the mathematically optimal solution'[81]. Heuristic analysis is used without the structure of a mathematical model for making decisions [81] and it is believed that it could assist in specifying the essential process for achieving the goal state [82]. It can also be used in a particular situation to specify the process involved in detecting an exception and taking corrective action [82]. Often, an heuristic algorithm is applied as it helps to provide a close right answer or solution for a specific instance of a problem [83].

2.7.1.2.2 Formal Analysis

"Formal analysis helps to detect many types of errors in a requirements specification either manually or automatically" [84]. Formal analysis uses formal notation which can be used to analyse and manipulate mathematical operators and mathematical proof procedures [84]. It also provides benefits in testing and proving the "*internal consistency including data conservation and syntactic correctness of the specification"*[85].

2.8 Related Work of Consistency/Inconsistency Management

2.8.1 Traceability

Many approaches have been proposed to maintain consistency and check inconsistency. One of them is traceability. The traceability technique is divided into two categories; forward/backward trace and derived. Olsson and Grundy developed a Web-based tool to summarise artifact data and to support basic explicit linking of elements in different representational models [75]. The method uses traceability and manages fuzzy relationships between high-level software artifacts (requirements), uses case models and black box test plans. The aim of this tool is to assist the inconsistency management for all changes made to artifacts. However, automation is impossible and that is needed to create a relationship. Further, "*high level natural language often lacks well-defined formal abstraction for all software artifacts representation"* [32].

Cysneiro and Zisman implemented the automatic generation of traceability relations among various types of models generated during the development of agent-oriented systems and identification of missing elements in the Prometheus model and JACK code specification [32] to check completeness in order to ensure the consistency between model and code specification is maintained, especially in a huge and complex system which involves different stakeholders. Rule- based approaches and Prometheus methodology are used with an extended version of XQuery to represent rules in traceability. However, this is still preliminary work and enhanced verification is needed.

Another technique to reduce inconsistencies among product lines was developed by Jirapanthong and Zisman [86]. XtraQue supports the generation of traceability relations in different types of documents that are capable of representing different levels of the development lifecycle of a product line [86]. It can define the semantics between the artifacts being compared and can also be used to bridge various activities and stakeholders taking part in the product line engineering [86]. It generates nine traceability relations such as satisfiable, ability, dependency, overlaps, evolutions, implements, refinements, containment, similar and different features based on OO documents created during development [86]. An extension of XQuery is used to represent the traceability rules and consider the semantics of documents, the traceability relation of various types of traceability with the product line domain and the grammatical roles of the words in textual parts of document, together with the synonyms and distance of words being compared [86]. A Rule-based approach is also applied to automatically generate the traceability relations among elements of documents that are created during the development of the product line system. Nevertheless, the "*existing rules failed to identify between requirements and object-oriented specification, besides changes in the documents require the traceability to be re-executed"* [86].

Goknil et al. [87] proposed an approach together with a tool for defining requirement relations using traceability. They cater for issues of consistency, change management and inference of requirements. First order logic is used to support the consistency checking of relations and to inferring new relations. However their approach only supports textual requirements and lacks consistency management between textual and other requirement artifacts such as use case and activity diagrams. There is also no automation provided for modeling requirements. The visualised result of either inferred relations or inconsistencies needs to be interpreted manually by the requirements engineer, which can lead to errors [87].

There is also a "*technique to recover traceability links between source code and free text documentation"* [88] using information retrieval which applies to both the IR techniques, namely as probabilistic, and vector space. This technique is applied to trace C++ and Java source classes to manual pages as well as the functional requirements. However, the effectiveness of this technique becomes less prominent when the number of familiar words between the source code component identifiers and the documentation item decreases [88].

2.8.2 Analysis Approach

2.8.2.1 Heuristic Analysis

As described earlier, analysis approaches are divided into two categories; heuristic analysis and formal analysis. The analysis approach is used together with requirement specifications and semantics. Heuristic analysis is one of the common techniques used to check for consistency or inconsistency of requirements. For example, Koth et al. [89] developed a technique to check the inconsistency of XML documents from a semantic point of view using an incremental attribute evaluation approach. This technique introduces incremental facilities and evaluates the attributes associated to XML semantics by "*adding an incremental strategy to XML semantic checker evaluator*" [89]. It also uses the Propagate algorithm and checks the consistency of documents repeatedly until a consistent document is produced. The efficiency of the evaluator is improved by lessening the re-evaluation process and evaluating the affected area only in the XML and not the entire document [89].

In addition, Chitchyan et al. implemented an automation support for requirements annotation, which is the extension of the WMatrix natural language processing tool suite called *RDL. "RDL is a tool enriched with the existing natural language requirements specification with semantic information derived from the semantics of the natural language itself"* [90] and MRAT tools (Multidimensional Requirements Analysis Tool) by Waters [91] which is an Eclipse plug in are used to facilitate the composition and analysis. MRAT is used to analyse the temporal relationship of RDL composition and it is believed that finding the temporal dependencies is useful to determine the points of sequencing conflicts and to avoid the conflicts and inconsistencies from happening.

Kroha et al. [92] investigated the use of semantic web technology to check the consistency of requirement specifications. They transform the static part of UML models that illustrate requirements into a problem ontology and attempt to discover inconsistencies by using ontological reasoning to uncover contradictions [92]. This work does not, however, check for behavioural consistency as it cannot represent dynamic aspects of UML specifications in the ontology.

Much research has been devoted to checking inconsistency and consistency using semi-formal specifications and heuristic algorithms. Egyed [26] implemented a UML-based transformation framework to check inconsistency and help in comparison. The author introduced an automated checking tool called VIEWINTEGRA which used consistent transformation to translate diagrams into interpretations and used the consistency comparison to compare those interpretations with those of other diagrams [26]. This technique can check inconsistencies without the help of third party or intermediate languages. The limitation of this tool exists when checking the consistency between an object diagram and state chart diagram or vice versa, as they cannot be transformed directly and need to be changed to a class diagram first in order to obtain the consistency results [26].

Sabetzadeh et al. [93] proposed a tool-supported approach for checking the consistency of a distributed model and enabling the checking for the inter-model properties of a set of models. This is done by checking of properties that merge within the intra-models [93]. A set of generic expressions is also developed to characterise the recurrent patterns in a structural constraint of a conceptual model. This approach currently works in the homogeneous model only as the merger cannot be defined at a notational level and this leads to a challenge in implementing this approach in a heterogeneous model [93].

In addition, Groher et al. presented an incremental consistency checker which allows one to define and redefine constraints [94]. This approach allows engineers to define and change the meta model and model any time "*without manual annotation or restriction on the constraint of the language used"*

[94]. This approach is implemented in a tool called "Model analyzer" which highlights the inconsistency of models in red [94].

Sinha et al. introduced a modeling environment called "Archetest" which adapts a unique bi-layer approach for precise modeling and automates the analysis [95]. It also analyses consistency and completeness. This approach accepts vague use case descriptions and helps to provide accuracy to them through a wizard-driven process [95]. However, more case studies are needed to test this work in order to prove its early results [95].

Kim [96] implemented a technique to assist the verification of user requirements expressed in natural language. This technique verifies the discrete event simulation model using a DEVS formalism together with a prototype tool called VERIDEV [96]. The verification consists of *"consistency verification between user requirements specification and class diagram, consistency verification between user requirement specification and sequence diagram of UML and the consistency verification between sequence diagram and DEVS diagram*" [96]. This technique is hard to apply because of difficulties faced in expressing it in the DEVS graph [96]. As a result, future enhancement is needed to automate the technique of integrity checking for the text base used in describing a model [96].

Chanda et al. proposed a formalisation methodology for the three most common uses of a UML diagram in capturing the static and dynamic aspects of an object oriented system: use case diagram, activity diagram and class diagram in order to emphasise inter-diagram consistency, syntactic correctness and traceability of requirement by using several formal rules [97]. A regular expression featuring (eg. +, *) is used to enhance the simplicity and understanding [97] of the grammar.

Jurack et al. presented a criteria for checking the consistency of refined activity diagrams which includes pre- and post- conditions [98]. The work used graph transformation rule sequences to define the behaviour of the refined activity diagrams to check consistency. This allows the analysis of a set of sequence to be conducted in a static manner [98]. However, the graph transformation rule cannot be checked with the static analysis and needs stronger reduction mechanisms to allow consistency analysis for a wider range of activity diagrams. For now, a restriction and assumption is applied to deal with the problem [98].

Litvak proposed an algorithmic approach to check the consistency of UML sequence diagrams and state diagrams [23]. An automation tool called BVUML is used to implement the consistency check algorithm [23]. The proposed algorithm helps in handling "*complex state diagrams such as fork, join and concurrent composite states*" [23]. The algorithm uses a breadth first search over the state diagram and a hybrid sequence state diagram is introduced to visualise the process with which the diagram state is associated to the sequence diagram [23]. This tool is sufficient for checking the consistency of a UML dynamic diagram, suitable for the standard UML and is demonstrated to be fast error detector. It is easy to use and does not require first or second order logic knowledge to generate or to understand the tool. In contrast, "*BVUML do not support purely syntactical states such as the stub states"* [23].

Whittle and Schumann presented an algorithm that works with a prototype tool in Java to automate the generation of a UML state-chart from a scenario in a form of sequence diagram [99]. Semantics information is added to the sequence diagram to detect and report inconsistencies [99]. The concept of hierarchy and structure in a form of class diagrams is used to show the merging of multiple sequence diagrams in a single state-chart [99]. However, the generated state-chart is just a skeleton and can be a substitute for manual refinement and modification [99].

Likewise, Li et al. [100]. have also conducted research into the consistency checking of UML diagrams. The research proposes a technique for checking the consistency of a UML requirement model which comprises use cases and conceptual class models with system constraint [100]. Together with this, the consistency of the requirements can be checked logically using semantics. However, this proposed technique only focuses on the aspect of formal model of requirement consistency. In order to validate the functional aspect of a requirement, a prototype generator tool is developed. It helps to automatically generate Java source code from the formal model of a requirement [100].

Zapata et al. detect consistency problems in UML diagrams by implementing a novel approach using Xpath and Xquery together with a rule-based system [101]. The reason for using them is because of "*their strange mix of suitability and standardization"* [101] that they can achieve. The main focus is to assess the "*consistency rules between UML class diagram and use case diagram"*[101]. These diagrams are integrated with OCL to avoid the ambiguities and guarantee the well-formed models in a formal way[101].

Alternatively, Satyajit et al. [20] suggested finding and specifying consistency conditions (CCs) for the domain in the initial abstract formal specification with the aim of recovering logical errors during the early phase of development. The RAISE Specification Language (RSL) [20] is used in writing the formal specification for this purpose. This tool combines the inspection of a specification and testing the executable specification of a prototype using test cases [20]. The intention is to validate the specification against requirements and to ensure the specified CCs are respected and maintained by the operation defined in the specification [20]. However, CCs are not used for checking the consistency of a requirement specification.

Blanc et al., proposed an approach to deal with inconsistency based on model construction operations, which uses logical constraints to define inconsistency rules and it is also meta model independent, which allows both intra-model and inter-model inconsistency rules to be defined and checked [102]. The consistency check is performed in a batch mode where the whole model is loaded into the memory and the verification starts by running the rules successively on the entire model [102].

Engels et al. presented a technique to specify and analyse consistency [103]. In order to conduct the checking process, models are mapped to semantic domains and behavioural constraints are analysed [103]. The problem of state-chart inheritance is demonstrated for this methodology [103]. A hybrid, rule-based notation is used. This rule combines textual styles of an attribute grammar with the queries of the meta model expressed as visual patterns [103]. The limitation faced by this technique is that it only supports partial resolution, although complete mapping is supported, and it needs tool support to generate a model compiler for the rule-based description provided [103].

Ha and Kang proposed several verification rules to check for consistency between UML static and dynamic diagrams such as class diagram, component diagram, state-chart diagram, sequence diagram, activity diagram, use case diagram, deployment diagram, collaboration diagram and object diagram [104]. A relation graph is used to show the relationship between diagrams. Consistency rules are developed from the relationships of both the object and dynamic diagrams [104]. However, these rules need help from the OCL (Object Constraint Language) as the rules need to first be transformed to formal language if the consistency checking is to be conducted automatically [104].

Ryndina et al. proposed a technique to establish the consistency between the business process model and object life cycles [105]. They defined two consistency notions for a process model called "*life cycle compliance and coverage"* [105] expressed in terms of conditions. A prototype tool that acts as an extension to the IBM WebSphere Business modeller was developed to help capture the existence of object states in the business process model, generating the life cycle from the process model and checking the consistency of consistency conditions [105]. This technique still has to be evaluated using a larger case study [105].

El-Attar and Miller proposed a structure presented in use case models called a Simple Structure Use Case Description (SSUCD) with tool support called SAREUCD which helps automate the detection and elimination of possible defects caused by inconsistencies [106]. The authors invented a technique called Reverse Engineering of Use Case Diagrams (REUCD) to generate use case diagrams from the SSUCD [106]. The SSUCD and REUCD processes allow the use diagram to be generated systematically and helps guarantee the "*consistency between the descriptions and their diagrams" [106].* However, the tool support still requires human intervention to fill in the details in each use case description before the tool can detect the inconsistencies. It also requires manual inspection for inconsistency if the segments are written in unstructured natural language [106].

Another approach is presented by Perrouin et al. for managing the inconsistencies amongst heterogeneous models by using a model composition mechanism [107]. The information of the heterogeneous models is translated to a set of model fragments [107]. Fusion is applied to build a global model which allows various inconsistencies to be detected, resulting in the global model [107]. Automation is applied to compute traceability links between the input model and the global one and thus supports the reporting of the inconsistencies on the original model and helps to resolve the cause of the inconsistencies [107]. However, the classification of which inconsistencies need to be resolved is not provided [107].

Mehner et al. proposed an approach for analysing the interaction and the possible inconsistencies that might exist in the requirement modeling phase [108]. A variant of UML with a use case-driven approach using use case diagrams, activity diagrams and class diagrams is applied [108]. The concept of pre- and post-condition using the UML variant of an activity is defined [108]. This requires more effort and it is recommended that there is an early formal analysis to overcome this problem [108]. The approach uses a formal technique called graph transformation with a tool support, AGG, in order to provide the chosen UML variant with formal semantics and allow a thorough and automatic analysis to be conducted [108]. This approach also allows the analysis of the interactions between the functional and non-functional aspects to be conducted automatically [108].

El-Mahded and Maibaum developed a tool called GOPSD to develop aspect-based process control and checking for the consistency and completeness of a requirement [109]. The tool adapts the concept of goal-driven analysis which was adapted from the KAOS technique for addressing process control systems [109]. The tool offers an animation utility which helps to reason about the taken actions in terms of "*aspect goals*, cycle by cycle and during the symbolic execution" [109]. Further evaluation is needed for these purposes. GOSPD also covers the early stage of development and it "*refines the abstract user's needs to functional and formal specification"* [109]. The tool also transforms the requirements automatically to B specification but the requirement needs to be corrected and validated first by the user before the transformation can be conducted [109].

In addition, Grundy et al. introduced a methodology of aspect-oriented component engineering to overcome problems related to component requirement engineering [110]. Their methodology analyses and characterises the component based on the "*different aspects of the overall application a component addresses"*[110]. The authors developed tool support which helps to specify aspects of a component in a component based software development environment [110]. The tool is equipped with basic validation checking in order to make sure all aspects of a requirement are met correctly [110]. This tool also provides a basic inconsistency management technique to help to manage the evolving aspect-oriented requirements including a highlighting of the change facility for all types of views and consistency checking via the matching of required links between components [110].

Another work related to checking the consistency using an aspect-oriented paradigm, this time for web applications, is by Yu [111]. The author presents a tool called HILA which was designed as an extension of UML state machines to model the adaptation rules for web application [111]. However, this work is believed not to be limited to web engineering applications only but may also be applicable to various other areas [111]. HILA could be helpful in improving the modularity of models and helps to automate the consistency checking of aspects to ensure rules are always in a consistent state [111]. However, HILA is likely to be useful to model the content and presentation only if it is modeled in a base machine [111].

To sum up, there are many approaches and techniques used to check or manage consistency and inconsistency of requirements by using heuristic analysis. Some of the techniques [23],[26],[92],[93],[95],[98],[99],[105] are well generated but most are still immature and need further enhancements. We have identified that most work needs tool support for the checking process. However, it is also true that, most work integrates with other available tools and are not purely built for consistency checking, especially when this needs to deal with processing natural language. Most tools or approaches do not support rigorous checking for consistency but only support partial solution for checking or identifying inconsistency and with a homogeneous model of a set of requirements. It is also identified that the tools developed still need human intervention to interpret the consistency results or invoke the action to check for the inconsistency although impressive and high level form of techniques are applied.

2.8.2.2 Formal Analysis

There are many researchers dedicated to checking the consistency of requirements using formal analysis together with the use of different types of requirement specifications and semantics. For example, Nenwitch et al. presented a lightweight framework called Xlinkit in order to check the consistency of distributed and heterogeneous documents using first order logic and lightweight mechanisms [29]. The main contribution of this framework is the definition of an extended semantics based on first order-logic and producing hyperlinks which diagnose inconsistencies across the specifications at different stages. The incremental checking technique used can also decrease checking time. However, XLinkit's limitation is that it lacks discovery of problems if the inconsistencies are recognised.

Other work by Nentwich et al. proposes a repair framework for inconsistent distributed documents [76]. They generate interactive repairs from an input of a first order logic formula that constrains the documents. Their repair system provides a correct repair action for each inconsistency together with available choices to handle the problem. However, they face problems when the repair actions interact with the grammar in a document, and also actions generated by other constraints [76]. Their approach also fails to identify a single inconsistency that may lead to other inconsistencies [76].

Other than that, Chen and Ghose developed an automated tool using the semantic web technology called SC-CHECK. This tool mainly focuses on the consistency management in distributed requirements engineering, especially in detecting inconsistency and "*supporting resolution in the context of industry-standard requirements specification notations"* [112]. Prolog is used to identify the violated consistency rule and possible errors or elements that need to be repaired. It uses an informal requirement specification and semi-formal representation in a form of sequence diagram for abstracting the formal representation to detect and resolve the inconsistencies [112]. It also provides the user with guidance to attempt to correct the inconsistencies. This tool has been tested via a medium scale case study and the results seem beneficial.

Zowghi et al. [17] proposed a technique to detect inconsistencies in requirements and the way to deal with them in a formal manner. A prototype tool named CARL is used to test the technique and it can perform an exhaustive search for plausible scenarios which cause latent inconsistencies to emerge [17]. A reasoning engine called CARET is applied to natural language requirements in order to analyse the inconsistencies within the translated logical statements of requirements. A simple engine for natural language parsing called *Cico* is used as a generic syntax-based parser by taking a subset of the English grammar as its domain [17]. It uses an application of the fuzzy rewriting system to a text and uses heuristic optimisation strategies and backtracking too. It is also believed that it could help in identifying and handling inconsistencies in Natural language requirements. Although it is useful in identifying, analysing and handling inconsistencies, a more expressive logic to specify more complex requirements is needed and, in order to hold the extended logic, the NL translation needs to be refined . Then, Gervasi and Zowghi [30] used the tool in detecting, analysing and handling inconsistencies in requirements for various stakeholders. It extends the tool with employing the theorem-proving and model- checking techniques in the context of default logic and it shows how to deal with the problems in a formal manner. The limitation of this tool is that the propositional logic used is not powerful enough to model adequate detail and accurate complex system behaviour. Propositional logic is not meant to detail the way the system should behave but is only suitable for high-level requirements [30].

Taibi et al. [113] implemented an algorithm for self-checking consistency for the classes using Object-Z specification. Verification utilises a test of specification, model abstraction and model checking. This algorithm conducts self-consistency only for each class and does not ensure consistency for the whole specification [113].

Kaneiwa and Satoh introduced an approach for conducting well-mannered consistency checking of UML class diagrams by translating the identified inconsistencies to first-order predicate logic [114]. They introduced an optimised algorithm with respect to the size of the class diagram to calculate *"the respective consistencies of class diagrams of different expressive powers in P, NP, PSPACE, or EXPTIME*" [114]. This work also helps to confirm the restrictions' existence for the class diagram in order to avoid any logical inconsistency.

Lamsweerde et al. [24] proposed a framework which is based on both formal and heuristic techniques to discover the conflicts and divergences of the goals or requirements of a domain property from a specification. The notion of boundary condition and domain knowledge plays an important role in this technique and the KAOS language is chosen as the specification language. Here, model checking is used to detect divergence among goal assertion [24]. It helps to capture the existence of different types of concept during the elaboration of requirements [24].

Kozlenkov and Zisman manage the consistency between natural language requirements and software artifacts that are generated during different phases of software system development, using a specific tool which embodies a goal driven and formal reasoning approach inclusive of "*goal elaboration, ordered abduction and morphing of path"* [74]. This is applied together with the use of knowledge-based and rule-based approaches. The weaknesses of this tool are that the inconsistencies discovered are limited to those related to the structure that has been recognised grammatically in natural language sentences only, and that the type of structures used needs to be expanded in order to allow the approach to be used in a large scale application [74].

Mu et al. [115] presented a merging-based approach to handling inconsistency by prioritising software requirements locally using the Viewpoints framework, which consists of requirement collection with local prioritisation. Then the "*requirement collection with local prioritization is transformed into stratified knowledge base"* [115]. The authors choose to use categorisation of the priority - High, Medium and Low. The first order logic is the best suited for this approach. Priority of requirements is measured to be a beneficial clue in resolving conflicts and making trade-off decisions. Conversely, there are problems that occur while presenting the merging process. In a few cases, the user may not obtain a stratified merge requirements collection and the introduced model-based merging operators could lead to difficulty in explaining the additional formulas in term of the viewpoint demands of the merge result [115].

Weitl et al. implemented an approach based on user support with the combination of pattern-based specification, temporal logic and ontology [116]. A Description Logic (DL) specification is used to represent the ontology and the content of the documents [116]. The verification framework is knowledge-based and the technique to support user specification is based on example- and a pattern-based approaches which are themselves based on the concrete examples of both correct and incorrect documents. This approach aims to check the consistency of high level structure with the content document and to check semantic consistency criteria on the context-dependent documents with high expressiveness, flexibility, applicability and high degree usability [116]. However, a broad knowledge of the logic used is needed in order to interpret or create the diagram and to apply the temporal formalism for solving the verification problems [116].

Scheffczyk et al. propose "*formalizing the temporal consistency rules and generating a few domain specific repairs for inconsistencies"* [117] of an industrial specification, using specification examples which focus on the functional requirements of a specification such as business processes, use cases and dialogues [117]. Therefore, a semi-formal consistency management toolkit called CDET is used to improve the quality of the industrial requirement specification. CDET can be used as a tool to "*check the semantics at different granularity levels and integrates fully with established practices"* [117] and daily project work. CDET is also integrated smoothly with the arbitrary revision control system (RCS) with the aim to establish a work process. CDET uses derivation of temporal predicate logic to facilitate consistency checking across the document revisions and it is suitable for checking any property of a document that is computable [117]. Although this tool is profitable for heterogeneous documents, experts in the field of logic are required to formalise the consistency rules [117].

Sousa et al. [53] presented an approach of using formal specification to check for inconsistency in a requirement. They used the B specification as a formal language derived from a controlled natural language [53] in the form of use case descriptions or scenarios together with the B method - a well-known formal method based on "*first order logic, a set of theory, integer arithmetic and generalized substitutions"* [53]. The work automates the analysis of requirement consistency against constraints (safety property) with the B method tool to reveal the inconsistencies in the specification. However, the work still lacks supports in terms of quality dimensions such as correctness and timeliness, lack of automatic consistency recovery such as a suggestion for changes and lack of support for a complex scenario and definition of grammar rules for use case scenarios and properties [53].

2.9 Analysis of Consistency / Inconsistency Management Research

From the discussed related work, we present a heat map in Figure 2.8 to show a categorization of the corpus work based on the type of specifications used, the type of contributions from all the researchers, type of semantics applied to each work and the type of techniques used to manage the consistency of requirements. The categorisation is mapped using colours (multiple tones of dark orange to light yellow) whereby mapping towards dark orange includes a higher percentage of papers following into this category.

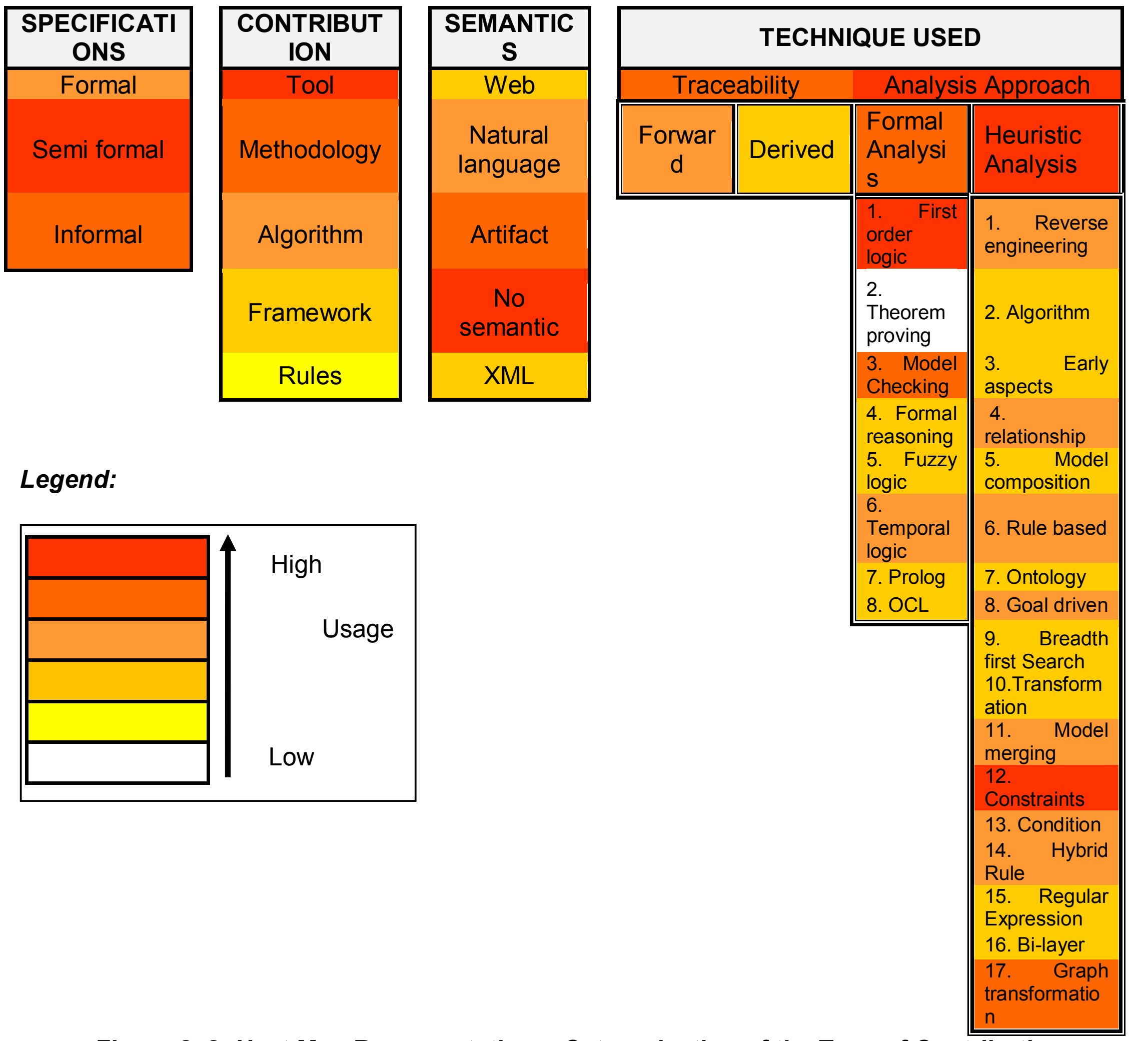


Figure 2. 8: Heat Map Representations: Categorisation of the Type of Contributions, Techniques, Specifications and Semantics Used for Checking the Consistency/Inconsistency

To sum up, there are three types of specifications used to represent the requirements in a form of formal, semi-formal or informal specification. The specification most often used is semi-formal. The types of semi-formal specification used are UML models, structured diagram, scenario/textual description, description logics and other components. Most of the semi-formal specifications used are UML models. A UML model is described as the design representation of the source code and this diagram is useful in making the source code understandable [118]. Semi-formal specifications also receive great interest here because the models are easily decomposed into smaller parts and this allows them to be better understood [57]. This feature encourages much works done to check the consistency between models although some studies concluded that maintaining consistency between models is not important but expensive [57]. Another specification used to represent requirements in performing consistency checking is informal specification written in natural language. The least used representation is formal specification. This is because the use of formal specification is challenging and hard for the beginner to use for fast results [55].

There are also five types of contributions for conducting consistency checking work; tool, methodology, algorithm, framework and rules. Tools gain most interest from researchers to perform the checking for inconsistency. This is followed by the development of methodology and algorithm. Quantitative evidence proves Yu's [50] point of view discussed in the previous section. The least research focuses on developing frameworks and rules to handle the consistency.

Most of the research to date has not applied any semantics. However, there are some works which applies the semantics of an artifact, followed by the semantics of natural language as well as the semantics of the web and of XML.

Techniques are categorised into two types; traceability and analysis approaches. The analysis approach is used more than traceability because the latter has been identified by several researchers as being complicated and costly to use and it is also seen to have no proper method to conduct [31], [5]. Further, the current automated approaches of traceability do not allow engineers to have proper means to visualise the result [119][33]. Both techniques are then divided into several sub-techniques. Traceability is divided into forward- and derived techniques. Here, the forward technique is used more than the derived to present the trace. The analysis approach is divided into formal analysis and heuristic analysis techniques. Figure 2.8 shows that, heuristic analysis is applied more by researchers than is formal analysis to perform consistency checking of requirements. For the formal analysis technique, first-order logic is chosen most by researchers, followed by model checking, temporal logic, formal reasoning, fuzzy logic, prolog and OCL. Theorem proving is least used in this work. For heuristic analysis, the use of constraints to perform the consistency checking has received enormous interest by researchers; followed in descending popularity by the use of graph transformation, reverse engineering, relationship, rule-based, goal driven, model merging, condition and hybrid rule. The least used techniques are algorithm, early aspects, model composition, ontology, breadth first search, transformation, regular expression and Bi-layer.

As shown in Figure 2.8, most researchers use semi-formal specifications to represent the requirements to check for inconsistency. To investigate in more detail the type of model used, we further classified the type of model using a heat map similar to the approach in Figure 2.8. This is shown in Figure 2.9. From the 43 works discussed, more than half used the semi-formal specification. Based on the classification of model used in a semi- formal specification approach in Figure 2.9, the most used model in consistency checking work is the UML model. The use case diagram and class diagram are used in most of the works, followed by the sequence diagram, state chart diagram, activity diagram and object diagram. Models other than the UML, such as Scenario or textual requirement description, are also used in checking consistency. Models such as task model, Essential Use Cases and Conventional Use Cases showed less and almost no interest by the researchers in checking the consistency of requirements, although Biddle et al. [43] found that Essential Use Cases open fruitful research of consistency issues between the responsibility concept in the requirements and their related designs as they help to improve the traceability support [43].

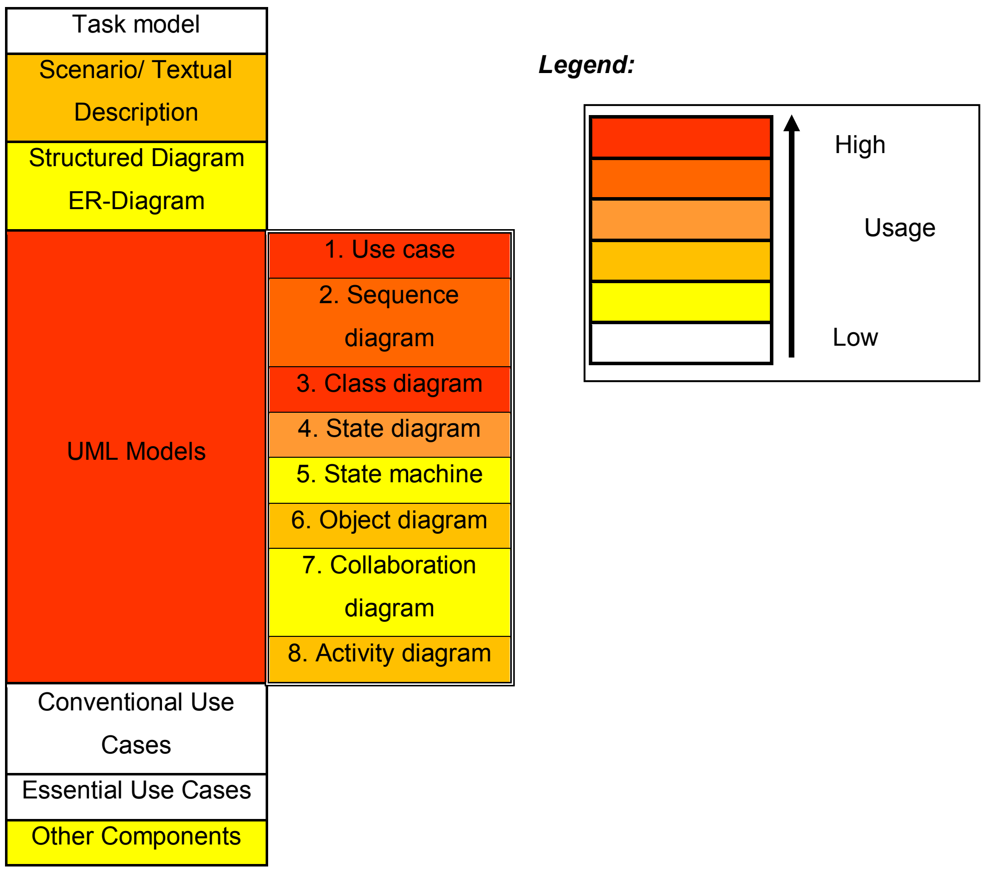


Figure 2. 9: Heat Map representation: Classification of the Model Used as a Semi-Formal Specification Approaches

2.10 Discussion

In this chapter, we provided a general overview of the requirement management process and requirement modelling techniques as well as the type of requirement specifications and requirement semantics. Then we discussed and categorised the requirement engineering tool, commonly called a requirement management tool, to identify the type of current RE tool and the processes it covers. The type of tool is classified as either a commercial or a research tool. The techniques and type of specification applied to each tool are also identified. These were presented in Table 2.1. The results show that most RE tools do not cover the validation process thoroughly. This led to a discussion of requirement validation in general, drilled down to consistency management. The different types of techniques used in consistency management were also discussed.

We conducted a literature review of related works based on the type of techniques used to manage consistency. From here we simplified and categorised the types of contributions, specifications, semantics and techniques used in the field of consistency management of requirements. We also compared the existing works and approaches and identified their strengths and weaknesses. We will use this to motivate us to form a basis for our research development outlined in following chapters.

A heat map was used to represent graphically the data of the types of contributions, specifications, semantics and techniques used in the consistency management simplified from the related works. Here, colours are used to show the frequency of the usage. The higher the value of usage, the darker the colour of the squares. A similar approach was applied to represent the type of models used as a semi-formal specification to represent the requirements.

The techniques used are categorised into traceability and analysis approaches. Each technique is then divided into smaller categories. Traceability is divided into forward- and derived techniques whilst analysis approach is divided into formal and heuristic analysis techniques. Each formal and heuristic analysis has its own sub-categories as described in previous sections. Most research uses a combination of techniques. The semi-formal specifications technique is widely used by most researchers. We then analysed the model used for the works, applying the semi-formal specification technique through the heat map representation in Figure 2.9. In addition, from the related works as well as the analysis, we found that traceability techniques are less often used for consistency checking work due to the difficulties mentioned. UML diagrams gain more interest by researchers in checking the consistency. Other models such as Essential Use Cases are not explored, although they are recognised as beneficial in checking the consistency of requirements and designs and have the ability to improve the traceability support. Further, most of the research does not have full coverage of the consistency checking of the requirements but tends to partial consistency checking, which focuses either only on the consistency of the natural language requirement or the models, or consistency between the natural language requirement and the models. Visual capability such as highlighting the inconsistency is less used to detect the inconsistency. Almost no research provides full end-to-end consistency checking support, which means from the natural language requirement to models and then to the prototype. Most research is mainly for the understanding and responsibility of requirement engineers and almost none support confirming consistency and validating requirements from the clients' side.

2.11 Summary

As described in the previous section, almost no research in managing consistency uses the Essential Use Case representation. Very little of the research discussed provides full end-to-end consistency checking support, which means from the natural language requirement to models and then to a prototype. Most of the work is also just concerned with verifying requirements by requirement engineers and not by the clients.

Therefore, the aim of the research presented in this thesis is to provide end-to-end support for checking the consistency of requirements in order to allow both the requirements engineer and the client to confirm and verify requirements consistency from an early stage of requirement analysis. We want to have a full coverage of inconsistency checking which is not limited to a partial solution or partial components to be checked.

We improve traceability by implementing a lightweight approach together with a traceability technique and semi-formal specification in the form of Essential Use Cases (EUC) models in order to support consistency checking between the natural language requirement, the EUC model and the prototype. As identified by Biddle et al. [43], EUC has merits in handling consistency issues and this led us to demonstrate that EUC provides benefits for inconsistency checking of requirements, although almost no previous researchers have used EUCs in their consistency checking work.

As a visual approach has not been well explored to detect or notify inconsistencies, we embed our approach with the visual capability provided by the Marama meta-tool [120] to highlight and notify to the user warnings regarding the existence of inconsistencies in any requirement component.

The following chapters will describe our approach to achieve our aims of providing end-to-end support for checking the consistency of requirements.

Chapter 3: Motivation and Overview of Our Approach

This chapter describes the motivation of our approach for this research on managing the consistency of requirements using a semi-formal specification in the form of an Essential Use Case (EUC) model with traceability management support. As mentioned in Chapter 2, we were motivated to apply EUC in this work as it provides a fruitful research area for consistency. In this chapter, we study further the usage of EUC in capturing and modelling requirements. We start with an introduction of EUC in semi-formally capturing a requirement from a textual user scenario, and then describe an experiment applying EUC within a group of postgraduate students. The results are analysed and motivated us to develop a lightweight and end-to-end approach to manage requirements consistency. An overview of our approach is also described.

3.1 Introduction

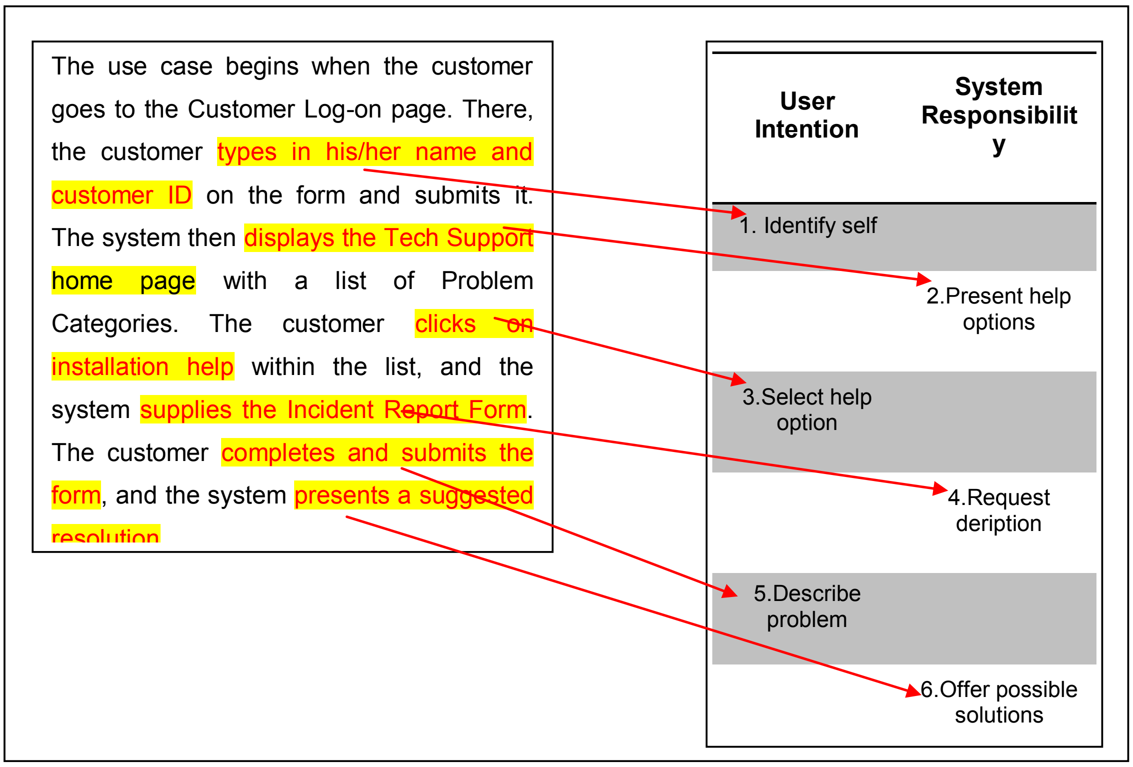
At present, when capturing software requirements from clients, requirements engineers often use some form of natural language, written either by clients or themselves. This forms a human-centric representation of the requirements accessible to both the engineer and client. However, due to both the ambiguities and complexities of natural language and the process of capture, these requirements often have inconsistencies, redundancy, incompleteness and omissions. Therefore, engineers often use models to represent these informally-expressed requirements which allow for better checking, analysis and structured representations, ideally leading to engineering higher quality systems.

There are many ways, identified in the previous chapter, to represent software requirements. Most common practices use some form of structured model. Models for our purpose can be defined as "*simplified representations of a complex reality and actually are forms of abstraction*" [121] where the act of abstraction is a "*process of focusing on those features that are essential for the task at hand and ignoring those that are not*" [121]. UML models are a common way of capturing software requirements [122] especially use case diagrams which are widely used by developers and requirements engineers to elicit and capture requirements. UML use cases capture functional requirements and, as applied in software engineering, deal with actor/system interaction [123]. Various studies have determined that eliciting requirements and extracting their use cases can be arduous and can lead to a rather imprecise analysis [39],[40],[41],[42]. Due to these deficiencies, Constantine and Lockwood [4, 123] were motivated to develop the Essential Use Case (EUC) modelling approach to overcome some of these problems. Although the usage of EUCs is not as widespread as conventional use cases, several researchers have recommended their adoption as their use helps to integrate the requirement engineering and interaction design processes [39],[43],[44]. Some of the main reasons why EUCs are not commonly used are because of a lack of tool support, engineers' lack of experience in extracting essential interactions from requirements and a lack of integration with other modelling approaches [39],[43].

3.2 Overview of Essential Use Cases (EUCs)

The EUC approach is defined by its creators, Constantine and Lockwood, as a "*structured narrative, expressed in a language of the application domain and of users, comprising a simplified, generalized, abstract, technology free and independent description of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intentions underlying the interaction*" [4]. An EUC takes the form of a dialogue between the user and the system. The aim is to support better communication between the developers and the stakeholders via a technology-free model and to assist better requirements capture. This is achieved by allowing only specific detail that is relevant to the intended design to be captured [43] . Compared to a conventional UML use case, an equivalent EUC description is generally shorter and simpler as it only comprises the essential steps (core requirements) of intrinsic user interest. It contains user intentions and system responsibilities to document the user/system interaction without the need to describe a user interface in detail. The abstractions used are more focused towards the steps of the use case rather than narrating the use case as a whole. A set of essential interactions between user and system are organised into an interaction sequence. Consequently, an EUC specifies the sequence of the abstract steps and captures the core part of the requirements [43]. Furthermore, the concept of responsibility in EUC aims to identify "*what the system must do to support the use case" without being concerned about "how it should be done*" [43]. By exploiting the EUC concept of responsibility, a fruitful research area on the consistency issues between responsibility concepts in requirements and their related designs is opened, which can potentially be used to improve traceability support. EUCs also benefit the development process as they fit a "*problem-oriented rather than solution–oriented*" approach and thus potentially allow the designers and implementers of the user interface to explore more possibilities [44]. They also allow more rapid development: by using EUCs, it is not necessary to design an actual user interface [43].

Figure 3.1 shows an example of a textual natural language requirement (left hand side) and an example Essential Use Case (right hand side) capturing this requirement (adapted from [123]). On the left is the textual natural language requirement from which important phrases are extracted (highlighted). From each of these, a specific key phrase (essential requirement) called an abstract interaction is abstracted and is shown in the Essential Use case on the right as user intentions and system responsibilities. This assists to abstract the requirements for specific technologies. For example, the requirement of typing in login information and using biometrics as an identification tool are transformed to a more abstract expression of requirement called "identify self".



The use case begins when the customer goes to the Customer Log-on page. There, the customer types in his/her name and customer ID on the form and submits it. The system then displays the Tech Support home page with a list of Problem Categories. The customer clicks on installation help within the list, and the system supplies the Incident Report Form. The customer completes and submits the form, and the system presents a suggested resolution

Figure 3.1 : (left) Example of Textual Natural Language Requirements and (right) Example of Essential Use Case (EUC model) [4],[123]

Although EUCs simplify captured requirements compared to conventional UML use cases, requirements engineers still face the problem of "finding the correct level of abstraction, which also takes time and effort" [39]. Requirements engineers need to abstract the essential requirements (using the EUC concept of abstract interactions) manually. This means understanding the natural language requirements and then extracting an appropriately abstract essential requirement embedded in a logical interaction sequence. To understand better the difficulty of achieving this, we conducted a user study of postgraduate students experienced in requirements elicitation and observed both their accuracy and time duration in undertaking Essential Use Case analyses manually.

3.3 Applying Essential Use Cases: A Study

Previous research of the EUC approach and practice in their use to model software requirements have indicated that requirements engineers sometimes have difficulties in identifying the "abstract interactions" used by EUCs and their sequencing [39]. This observation, while intuitive, is anecdotal. To obtain a better understanding of these difficulties, we conducted a user study of several requirements engineers carrying out the extraction of an EUC model from a set of requirements specified in natural language, in order to observe their performances and experiences in using EUCs. We used the same requirements as described in [123] and compared the abstracted EUCs in that work to the results developed by our EUC model developers.

The study participants were 11 post-graduate software engineering students, several of whom had previously worked in the industry as developers and/or requirements engineers. All were familiar with UML use case modelling and most had previously used UML use cases to model requirements. None were familiar with the EUC modelling approach. Each participant was given a brief tutorial on the EUC approach and some examples of textual natural language requirements and derived EUC models. Participants were asked to develop an EUC model from the textual natural language requirements and we tracked the time taken and analysed the accuracy of their resulting models. We gave them Constantine and Lockwood's "getting cash" scenario (as shown in figure 3.2), which we have slightly refined, to analyse. The small refinement was to add a sentence (sentence number 11 from Figure 3.2) to improve the clarity of the scenario to the participants. This is a common template of user/system interactions common in many web-based systems as well as ATMs and other kiosk-like systems. Intuitively, the extraction of a set of essential user/system interactions from this example to form an Essential Use Case structured model of the requirements should be straightforward.

1. The use case begins when the Client inserts an ATM card. The system reads and validates the information on the card.
2. System prompts for PIN. The client enters PIN. The system validates the PIN.
3. System asks which operation the client wishes to perform. Client selects "Cash withdrawal."
4. System requests amount. Client enters amount.
5. System requests type. Client selects account type (checking, saving, credit).
6. The system communicates with the ATM network to validate account ID, PIN and availability of the amount requested.
7. The system asks the client whether he or she wants receipt. This step is performed only if there is paper left to print the receipt.
8. System asks the client to withdraw the card. Client withdraws card. (This is a security measure to ensure that clients do not leave their cards in the machine.)
9. System dispenses the requested amount of cash.
10. System prints receipt.
11. Client receive cash.
12. The use case ends.

Figure 3. 2: The scenario "Getting Cash" Refined and Adapted from [4] Used For the Evaluation

Candida te	Answers												Time taken (minute s)
	Identify user		Verify identity		Offer choices		Choose		Dispense cash		Take cash		
1		X		X		X	Y		Y		Y		9
2	Y			X	Y		Y		Y			X	5
3		X		X	Y			X	Y			X	10
4		X		X		X	Y		Y			X	7
5		X	Y			X		X	Y			X	10
6		X		X		X	Y		Y			X	7
7	**Y**		**Y**		**Y**		**Y**		**Y**		**Y**		**20**
8	Y			X		X	Y		Y			X	10
9	Y		Y		Y			X		X		X	10
10.		X		X		X		X	Y			X	25
11.	Y		Y			X		X		X	Y		10
Total	**5**	**6**	**4**	**7**	**4**	**7**	**6**	**5**	**9**	**2**	**3**	**8**	**123**
Average time:123/11=11.2													

Table 3. 1: EUC Extraction Study Results

Table 3.1 summarises the results of our study. The correctness (Y for correct, x for incorrect) and time taken were recorded for each person. A correct answer (Y) means that the answer provided by the participant is the same or very similar to the interaction pattern provided by a library pattern that we obtained from Constantine and Lockwood [4]. Summarising these results:

1. The number of correct interactions that identified (Y) = 31 out of 66 total correct interactions or 47% (i.e. 53% were incorrect).
2. The number of completely correct EUC interactions (all Ys) = 1 out of 11 or 9.1%
3. The average time taken to accomplish the EUC development task was 11.2 minutes. The longest time taken was about 25 minutes and the shortest time taken was about 5 minutes, so there was significant variation in the time taken.

Based on these results, participants were more likely to generate incorrect EUC interactions than correct ones, and very unlikely (9.1%) to produce a completely correct EUC. All but one participant failed to identify some of the essential interactions present in the natural language requirements; many failed to assemble these into an appropriate interaction sequence, and only one (participant 7, highlighted in orange in Table 3.1) managed to obtain a solution which was the same as or very similar to the model answer of the "getting cash" scenario of Constantine and Lockwood. The root cause of most problems was that participants tended to incorrectly determine the required level of abstraction for their essential interactions (the user intentions and system responsibilities of the EUC model). This is based on observation made as they performed the task as well as analysis of the incorrect answers provided by them. The study also demonstrates that it was quite time consuming for participants as they needed to figure out the appropriate keywords that describe each abstract interaction and to organise them into an appropriate sequence of user intentions and system responsibilities. We can see that there is a considerable variation in the time taken and also that the longest time taken does not ensure the correctness of the answer. For example the participant (participant 10, highlighted in blue in Table 3.1) who took the longest time (25 minutes) to accomplish the task only provided 1 correct essential interaction characterisation out of 6, a poor result; while one of the better participants (participant 2, highlighted in yellow in Table 3.1) took only 5 minutes to produce 4 out 6 correct interactions. Our survey thus supports the anecdotal findings reported by Biddle et al. [39] with more quantitative evidence.

3.4 Overview of Our Approach

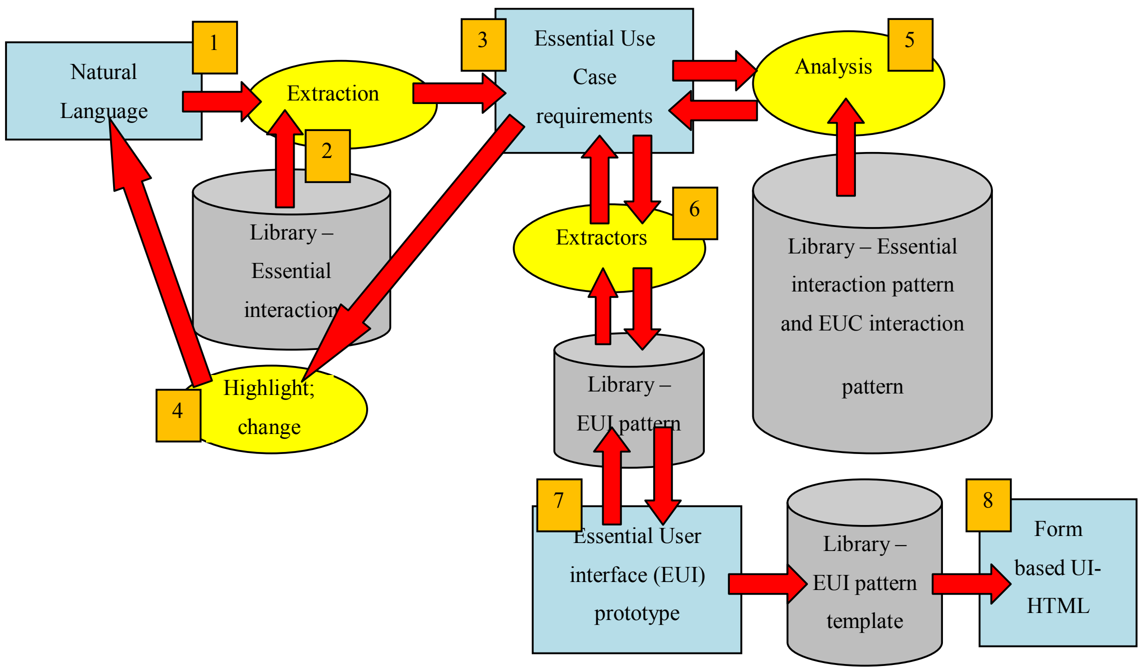
We were quite surprised by the results in the previous section. Many of the participants were experienced in the industry; they were academic requirements modellers and all were familiar with and most were experienced in using UML use case modelling. Given this background, we expected much more accurate modelling of the example requirements using the EUC technique. This study, while being quite small in nature, does support previous claims about the challenges in extracting natural language requirements into EUC models [39]. We then studied and evaluated a variety of of requirements collections in the form of user scenarios or use case descriptions amenable to modelling as EUCs. The requirements are derived from requirements engineering and software engineering books, published literature and published information from software developers web pages as well as some requirements collected from real requirements engineers and business analysts. From there, we come across various key phrases (essential interactions) for particular abstract interactions. This has provided us with the motivation to develop an approach and supporting tool which enables requirements engineers to extract accurate EUC abstract interactions automatically from textual natural language requirements. This is supported by related literature supporting the need for having an automated tool, We also speculated that our approach could provide better support to users and developers to work with informal and semi-formal requirements and keep them consistent. We have developed an automated prototype tool providing authoring facilities for textual requirements and for checking the consistency of these requirements. This tool assists requirement engineers and business analysts to check whether their requirements that are written or collected in natural language are consistent with other analysis and design representations.

We have implemented end-to-end support for consistency checking, using:

the 1) EUC modelling [123], 2) a high level user interface design in the form of low-fidelity prototype-Essential User Interface (EUI) prototype, 3) a concrete UI view in the form of a form-based UI (HTML page) as our semi-formal models. This was due firstly to their appeal as representations [10] that developers and end users could work with and secondly to limited research done to date investigating consistency issues with these representations and natural language requirement [43]. In addition, EUI pattern support is developed to allow reusability of the UI component and to enhance the accuracy of mapping the EUC model to the EUI prototype.

In order to support this concept, we have developed a traceability technique: this allows the elements of the textual natural language requirements written in a form of user-scenarios and Essential Use Case requirements, and UI prototypes to be kept consistent with one another. We believe that supporting this traceability will allow us to better detect and manage inter-specification inconsistencies and also enable developers and users to work more effectively with different models of requirements. We embed our consistency management and tracing tool within the Eclipse-based Marama meta-tool environment [120] in order to help to provide visual capability in detecting any inconsistency.

To support requirements analysis in improving requirements completeness and quality, complementary work is done in the collection and categorisation of terminology from different case studies and scenarios. For now, we have more that 15 scenarios and case studies which contribute to nearly 360 patterns of essential interactions and almost 80 patterns of abstract interactions. These are discussed in detail in the next chapter. This provides a library of essential interaction patterns and EUC interaction patterns which are reusable. They are also able to support various domains and assist engineers in finding appropriate abstract interactions for designing the EUCs for a system. A visual differencing approach is applied together with the essential interactions and EUC interaction patterns to improve the requirements quality such as the completeness and correctness. Figure 3.3 shows an example of our proposed approach.



Natural Language
1
Extraction
2
Library – Essential interaction
3
Essential Use Case requirements
4
Highlight; change
5
Analysis
6
Extractors
Library – Essential interaction pattern and EUC interaction pattern
Library – EUI pattern
7
Essential User interface (EUI) prototype
Library – EUI pattern template
8
Form based UI-HTML

Figure 3. 3: Overview of Our Requirements Consistency with End-to-End Support using EUC and a Traceability Management Approach

The processes of the outline approach are:

I. Firstly, textual natural language requirements in a form of user scenario of a use case usage (1) are analysed using a database of essential interactions (2). Phrases from textual natural language requirement are analysed and matched with the essential interaction pattern library to find an appropriate abstract interaction. For example; the phrases "display error" and "display incorrect error page" are mapped to abstract interaction "display error".

II. Then, Essential Use Case (EUC) Models are generated (3). A list of abstract interactions provided by the essential interaction pattern library is then mapped to EUC using the mapping engine to categorize each abstract interaction as either a user intention or system responsibility. Examples of related abstract interaction and two categories are shown in table 3.2 below. Each abstract interaction is then organised in a sequence of interactions as an EUC as shown in table 3.3.

Abstract interaction	**Category**
Request identification	System responsibility
Identify self	User intention

Table 3. 2: Example of Abstract Interaction and its Related Category

User intention	System responsibility
	1. Request identification
2. Identify self	

Table 3. 3: Example of EUC Model Generated

III. The user may select any abstract interaction of the EUC and see the originating textual natural language elements (4). The user, a requirements engineer or end user, may also change elements in the EUC model or textual natural language or the list of abstract interaction requirements and see the impact of the change in the other model (4).

IV. An analysis tool (5) uses a set of essential interactions and EUC interaction patterns to determine if an extracted EUC model is complete, consistent and correct according to acceptable patterns of essential interactions in the essential interaction pattern library and EUC interaction pattern library. A generated EUC model generated will be compared with the available "best-practice" template based on the specific domains of EUC interactions stored in the library.

V. Further extractors (6) map the EUC model to a low-fidelity prototype called as Essential User Interface (EUI) prototype, which is also a high level form of design (7). This is derived from the Essential Use Case requirements model with the support of the EUI pattern library. The EUI prototype can also be transformed to a concrete UI view as an HTML form-based UI (8) with the support of the EUI pattern template library. Some examples from the EUI pattern library are shown in Table 3.4. Overall, the approach supports end-to-end rapid prototyping with traceability and inter-model change management between textual natural language requirements and EUC models (4) and the EUI prototype.

EUI pattern	EUI pattern category	Abstract interaction of EUC model
List of options	List	Choose
		offer choice
		select option
Display payment	Display	validate payment
		show payment
ID	Input	identify self
		request identification
Help	Action	Ask help
		present solution

Table 3. 4: Example of EUI Pattern Library for EUI prototype

3.5 Summary

We have described the use of Essential Use Case models in capturing and checking the inconsistency of requirements. Problems such as lack of tool support, engineers' lack of experience in extracting essential interactions from requirements, and a lack of integration with other modelling approaches [39],[43] have motivated us to come up with a lightweight approach and tool support to extract the textual natural language requirement to a semi-formal model - an Essential Use Case. An essential interaction library is produced to support this process. This initial step in our goal of providing end-to-end support for requirement consistency is described in more detail in the next chapter.

Chapter 4: Essential Interaction Extraction

This chapter describes in detail our approach to capturing natural language requirements. It extends the previous chapter to describe the extraction of the essential interactions approach and the essential interaction library that we used to enhance the accuracy of essential requirements (abstract interactions) for the EUC model.

4.1 Introduction

A number of studies have shown the difficulty of using a heavyweight Natural Language Processing (NLP) tool which includes the use of sentiment and semantic analysis, parser, complex classifier and complex analysis of NLP and formal method techniques in dealing with and analysing natural language requirements. They have also shown that the results of such approaches are often imprecise and inconsistent [124],[36],[34]. Consequently we decided NOT to use any such approach to extract the essential interaction. Instead of using conventional NLP-based approaches, we adopted a more domain-specific approach. Extracting EUC essential interactions from textual natural language requirements constrains the problem domain to a set of suitable interaction descriptions. This means we chose to develop a library of "proven" essential interactions expressed as textual phrases, phrase variants and limited regular expressions. This library of essential interactions contains abstract interaction patterns that were developed from a collection of such patterns previously identified by Constantine and Lockwood [4] and Biddle et al. [39] together with patterns that were developed by us, which are all applicable across various domains.

Each essential interaction pattern in the library was also associated with a collection of alternative sequences of textual natural language requirement phrases that could match the pattern. Each of these sequences relates to a more concrete version of the abstract interaction pattern. The textual natural language requirements were then analysed by matching them against the concrete versions, looking for a good match. Abstraction can then be undertaken by creating an instance of the more abstract interaction pattern associated with the concrete one. The matching process used is similar to the process of keyword searching. Collectively, this provides a more lightweight approach to analyse the natural language requirements than NLP approaches, which is thus able to provide a set of meaningful abstract interactions to the requirement engineer. The abstract interaction patterns can be added in order to improve our ability to recognise essential interactions in textual natural language requirements. We can also segment the library into different patterns for different application domains as patterns are also commonly used for expressing reusable design. By using the patterns, the user will be more likely to get the outcomes right and so to acquire sensible EUCs. This contrasts with the results from the preliminary study reported in the Chapter 3, where most users tended to provide wrong answers rather than right answers.

After extracting a set of candidate essential interaction phrases and assembling them into a candidate sequence of abstract interactions, the requirement engineer is presented with a list of interactions with the original textual natural requirements juxtaposed on the screen. The engineer can then selects abstract interactions and see from where the textual natural language requirements were derived, or vice versa. The engineer can move interactions and add or delete interactions. A limited update of the textual natural language requirement is also supported. The engineer can modify the natural textual natural language requirement and see the impact on the re-extracted essential interactions. An Essential Use Case visualisation is also provided, conforming to Constantine and

Lockwood's [4] approach. It can also be edited with a limited update of the essential interactions from which it was derived and consequently the textual natural language requirement phrases. An update of the textual natural language requirement results in an update of the extracted essential interactions and Essential Use Case models.

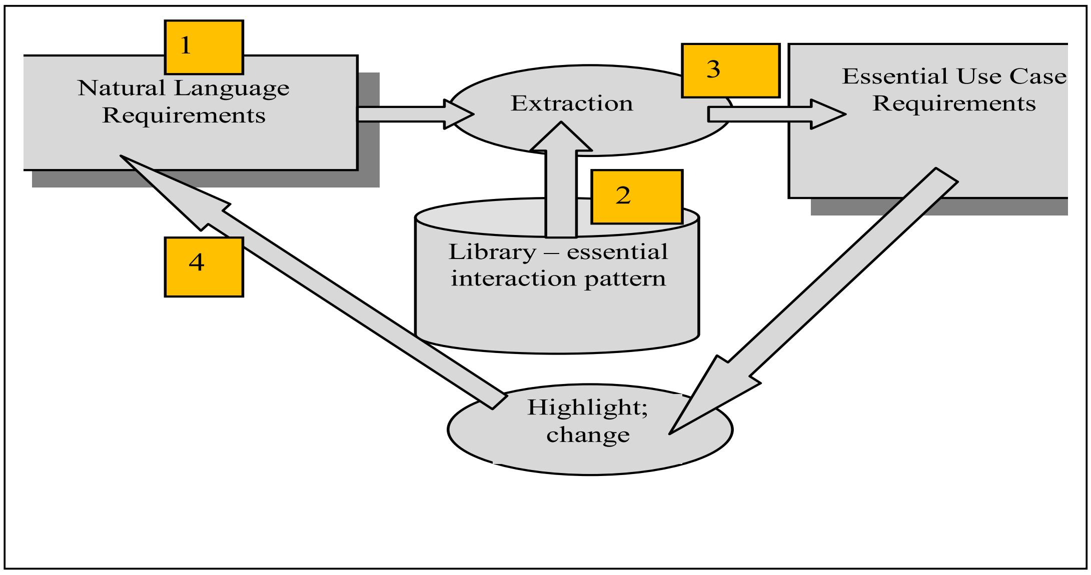

1
Natural Language Requirements
Extraction
3
Essential Use Case Requirements
2
Library – essential interaction pattern
4
Highlight; change

Figure 4. 1: Our Essential interaction extraction approach

Figure 4.1 illustrates this extraction/trace-forward/trace-back process that we provide to requirements engineers.

I. Textual natural language expressed requirements (1) are fed through an extraction process (2) which uses a library of essential interaction phrases and expressions, producing a sequence of EUC essential requirements.

II. The engineer can select items in the textual natural language requirements of EUC interactions (3) and see corresponding items (4).

4.2 Essential Interaction Pattern Library

In order to facilitate the extraction process, we have developed an essential interaction pattern library for storing all the essential interactions and abstract interactions. The essential interaction pattern library is based on a collection of phrases that illustrate the function or behaviour of a system. The collection of phrases is then categorised, based on its related or associated abstract interaction. We have collected and categorised phrases from a wide variety of textual natural language requirements documents available to us and stored them as essential interactions. Currently, we have collected approximately over 360 phrases from various requirement domains including online booking, online banking, mobile systems related to making and receiving calls, online election systems, online business, online registration and e-commerce.. The collection and categorisation of the phrases is an on-going process. Based on these phrases, we have come up with close to 80 patterns of abstract interaction. On average, there are 4.5 phrases or essential interactions associated with each abstract interaction. For example the abstract interaction "display error' is associated with four different essential interactions: "display time out", "show error", "display error message" and "show problem list". The essential interactions were not categorised based on one scenario. They have associations with five different concrete scenarios such as online business, e-commerce, online booking, online banking and an online voting system. This example shows that one particular abstract interaction can be associated with multiple concrete scenarios. Table 4.1, below, shows other examples of abstract interaction and its associated essential interactions for various domains of application.

Abstract interaction	Essential interaction	Example of Domains
Verify user	verify customer credential	Online banking, online booking, online business, e-commerce, online reservation
	verify customer id	Online banking, online booking, online business, e-commerce, online reservation
	verify username	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	check the username	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	check the password	Online banking, online booking, online business, e-commerce, online voting system, online reservation
Ask help	help desk	Online banking, online booking, online business, e-commerce, online reservation
	request for help	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	ask for help	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	clicks help	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	complete help form	Online banking, online booking, online business, e-commerce, online voting system, online reservation
Offer choice	prompt for amount	Online booking, online banking, online business, e-commerce
	display account menu	Online banking
	display transaction menu	Online banking
	display select function	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	display menu	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation

Table 4. 1: Example of Abstract Interactions and their Associated Essential Interaction and Their Related Domains

In order to store the essential interactions in the essential interaction pattern library, selected phrases ("key textual structures") are extracted from the textual natural language requirement, based on their sentence structure. The 'key textual structure" uses Verb-Phrases (VP) and Noun-Phrases (NP) in the sentence structures to categorise the essential interactions. Any phrases that follow this structure will be acceptable as an essential interaction in the essential interaction pattern library. The tree structure of the key textual structure is illustrated in Figure 4.2.

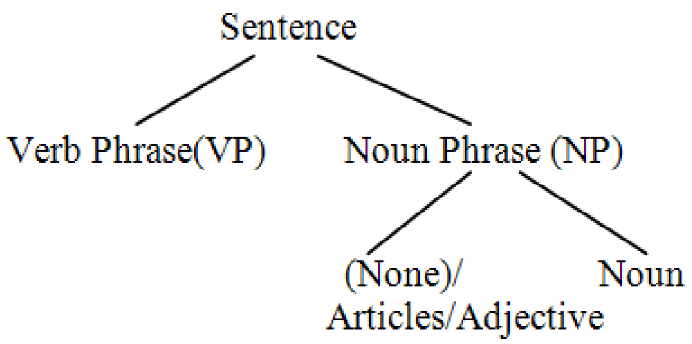


Figure 4. 2: Tree Structure for Key Textual Phrase

The tree structure in Figure 4.2 shows that our library has three different sentence structures, based on the location of the Verb Phrase (VP) and Noun Phrase (NP). The Noun Phrase can contain structure elements such as Articles (ART) and Adjectives (ADJ) or only Nouns (Noun).

The three different sentence structures are;

I. **Verb (V) + Noun (N) (only)** e.g. request (V) amount (N)
II. **Verb (V) + Articles (ART)+ Noun (N)** e.g. issue (V) a (ART) receipt (N)
III. **Verb (V) + Adjective (ADJ)+ Noun (N)** e.g. ask (V) which (ADJ) operation (N)

This key textual structure aims to provide flexibility in the library's ability to accommodate various types of sentences containing abstract interactions. With this, a broad range of phrase options can be extracted by the tracing engine, while still affording a lightweight implementation using string manipulation and some regular expression matching. Some examples of phrases stored in the essential interaction pattern library following the key textual structure are shown in Table 4.2. For now, we have performed this essential interaction pattern library development manually and plan for an automated approach in the future.

phrases	abstract interaction
identifies which item	select item
view the order details	view detail
creates a receipt	print
request for help	ask help

Table 4. 2: Example of Essential Interaction and its Associated Abstract Interaction stored in the Essential Interaction Pattern Library

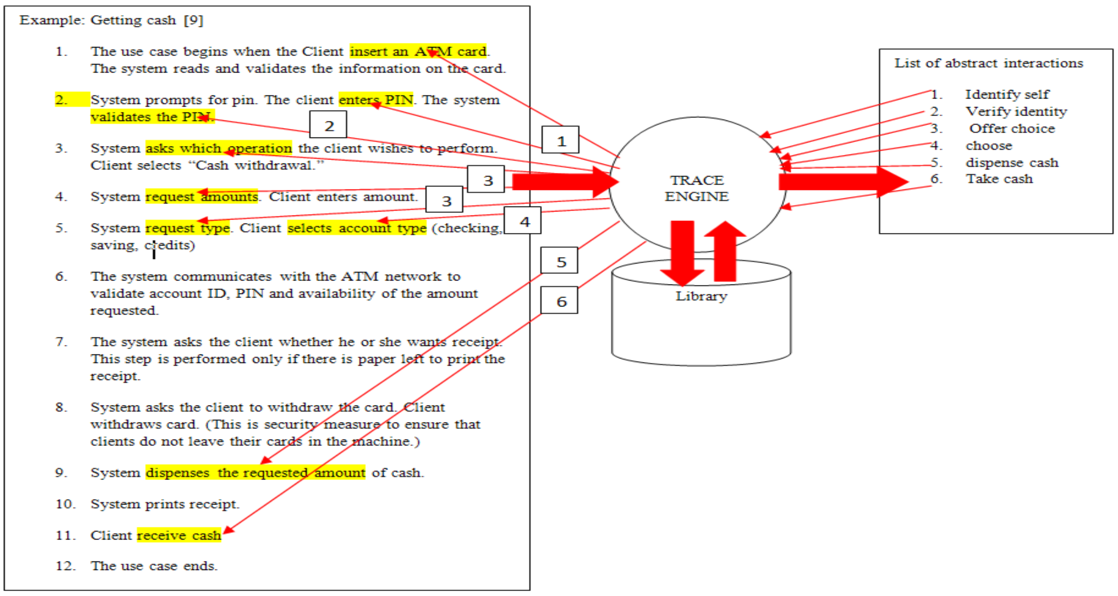
4.3 Tool Support

We have developed a prototype EUC essential interaction extraction tool based on the approach outlined in the previous section. The idea is for requirement engineers to use the tool to do an initial essential interaction extraction from textual natural language requirements, producing an initial EUC model. Selecting phrases in the textual requirements shows the resulting extracted essential interactions. Selecting essential interaction(s) shows the textual natural language phrase(s) the essential interactions were derived from. This provides a traceability support mechanism between textual natural language requirements and derived EUC models.

The engineer can then modify the resultant EUC model and/or the original textual natural language requirements. This includes adding phrases and interactions, re-ordering phrases and interactions, deleting phrases and interactions and modifying phrases and interactions' descriptive texts. The engineer then re-extracts the essential interactions and associated traceability links. Engineers can add new essential interaction phrases to their library or even develop different essential interaction libraries for different problem domains. The former allows our tool to improve its extraction support for users over time and the latter allows specific domain interaction patterns to be used. Guidelines for using the tool and the patterns are also provided. Engineers do need to have an understanding of the Essential Use Case concept and methodology before using the tool.

4.3.1 Tool Process

The framework for extraction, trace-forward and trace-back between the abstract interactions from the textual natural language requirements and vice versa is illustrated in Figure 4.3. We use the scenario of "getting cash", a similar scenario illustrated in the previous chapter as an example of extracting textual natural language requirements to Essential Use Cases. The tracing engine searches for key textual phrases (typically verb-noun phrases, such as "withdraw cash" or "request amount") contained in the library within the textual requirements. Having identified such matching phrases, it looks for orderings of these within the requirements that match orderings in the library associated with particular EUC interaction specifications. For example, in Figure 4.3 (1), the phrases “insert an ATM card” and “client enters PIN” are both associated, in that order, with the “identify self” abstract interaction. Having identified such essential interactions, the tracing engine instantiates the abstract interaction into the EUC model (to the right in Figure 4.3) and associates it with the identified key phrases in the textual requirements. This association allows trace-forward or trace-back to be supported with appropriate matching elements, highlighted in the other view when key phrases or abstract interactions are selected. This not only supports traceability between textual natural language and EUC model elements but also assists engineers and clients in understanding the quality of the requirements. For example, phrases with missing interactions and incomplete interaction sequences can be seen, interactions or interaction sequences with incomplete textual phrases or ordering/structure in natural language identified, and EUC models with inconsistencies or incompleteness, such as missing system responses to user requests, highlighted.



Example: Getting cash [9]

1. The use case begins when the Client insert an ATM card. The system reads and validates the information on the card.
2. System prompts for pin. The client enters PIN. The system validates the PIN.
3. System asks which operation the client wishes to perform. Client selects “Cash withdrawal.”
4. System request amounts. Client enters amount.
5. System request type. Client selects account type (checking, saving, credits)
6. The system communicates with the ATM network to validate account ID, PIN and availability of the amount requested.
7. The system asks the client whether he or she wants receipt. This step is performed only if there is paper left to print the receipt.
8. System asks the client to withdraw the card. Client withdraws card. (This is security measure to ensure that clients do not leave their cards in the machine.)
9. System dispenses the requested amount of cash.
10. System prints receipt.
11. Client receive cash
12. The use case ends.

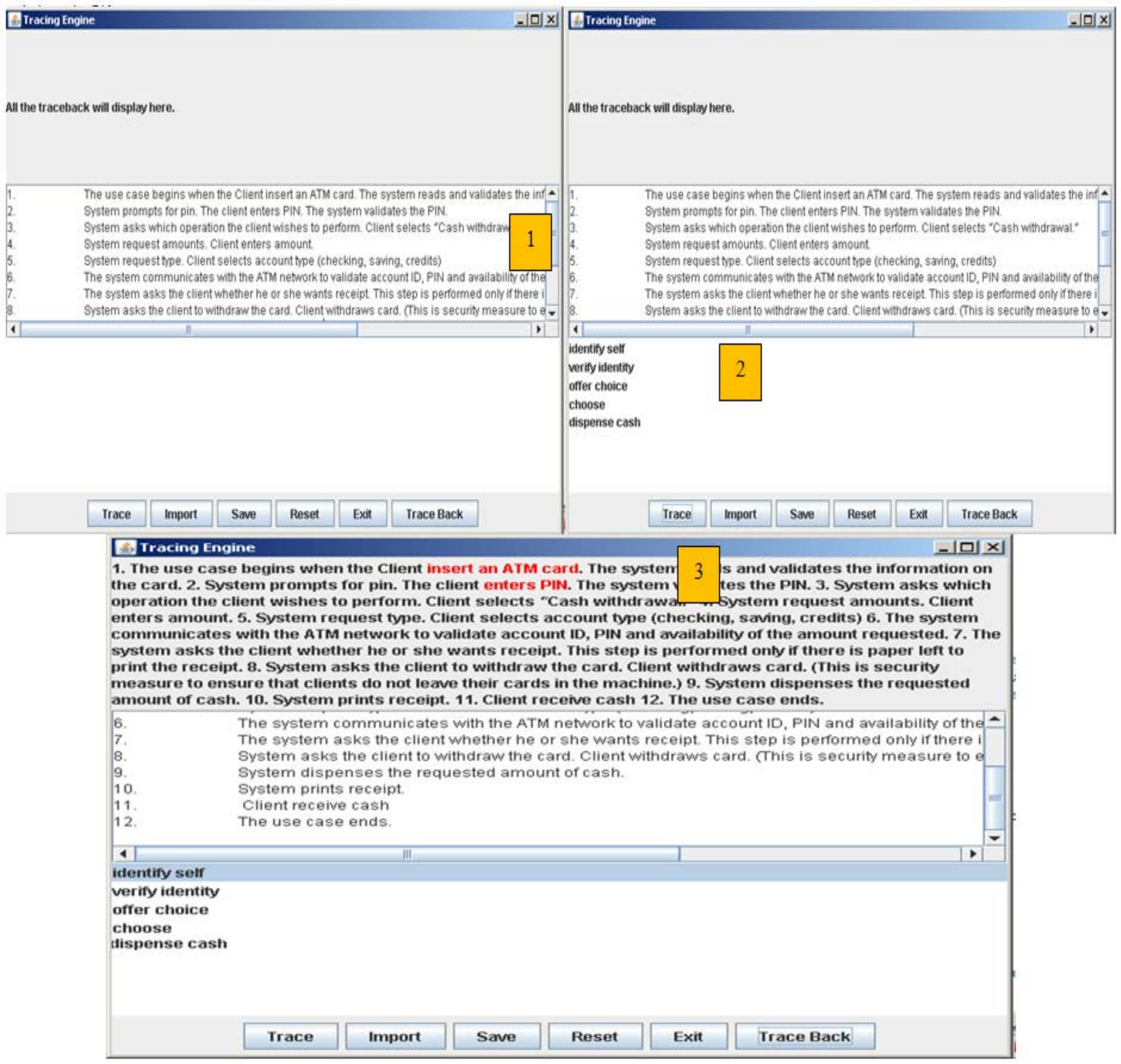
List of abstract interactions

1. Identify self
2. Verify identity
3. Offer choice
4. choose
5. dispense cash
6. Take cash

Figure 4. 3: An example of performing an essential interaction extraction to a EUC model and supporting trace-forward/trace-back

4.3.2 Tool Example

We have developed a prototype automated extraction and tracing tool in order to reduce the time taken to generate abstract interactions and increase the correctness level of each specific abstract interaction. Several screen dumps of the tool in use are shown in Figure 4.4.



```
Tracing Engine

All the traceback will display here.

1.          The use case begins when the Client insert an ATM card. The system reads and validates the inf
2.          System prompts for pin. The client enters PIN. The system validates the PIN.
3.          System asks which operation the client wishes to perform. Client selects "Cash withdraw
4.          System request amounts. Client enters amount.
5.          System request type. Client selects account type (checking, saving, credits)
6.          The system communicates with the ATM network to validate account ID, PIN and availability of the
7.          The system asks the client whether he or she wants receipt. This step is performed only if there i
8.          System asks the client to withdraw the card. Client withdraws card. (This is security measure to e
```

```
Tracing Engine

All the traceback will display here.

1.          The use case begins when the Client insert an ATM card. The system reads and validates the inf
2.          System prompts for pin. The client enters PIN. The system validates the PIN.
3.          System asks which operation the client wishes to perform. Client selects "Cash withdrawal."
4.          System request amounts. Client enters amount.
5.          System request type. Client selects account type (checking, saving, credits)
6.          The system communicates with the ATM network to validate account ID, PIN and availability of the
7.          The system asks the client whether he or she wants receipt. This step is performed only if there i
8.          System asks the client to withdraw the card. Client withdraws card. (This is security measure to e
identify self
verify identity
offer choice
choose
dispense cash
```

```
Tracing Engine
1. The use case begins when the Client insert an ATM card. The system [illegible]s and validates the information on
the card. 2. System prompts for pin. The client enters PIN. The system v[illegible]tes the PIN. 3. System asks which
operation the client wishes to perform. Client selects "Cash withdrawa[illegible]System request amounts. Client
enters amount. 5. System request type. Client selects account type (checking, saving, credits) 6. The system
communicates with the ATM network to validate account ID, PIN and availability of the amount requested. 7. The
system asks the client whether he or she wants receipt. This step is performed only if there is paper left to
print the receipt. 8. System asks the client to withdraw the card. Client withdraws card. (This is security
measure to ensure that clients do not leave their cards in the machine.) 9. System dispenses the requested
amount of cash. 10. System prints receipt. 11. Client receive cash 12. The use case ends.
6.          The system communicates with the ATM network to validate account ID, PIN and availability of the
7.          The system asks the client whether he or she wants receipt. This step is performed only if there i
8.          System asks the client to withdraw the card. Client withdraws card. (This is security measure to e
9.          System dispenses the requested amount of cash.
10.         System prints receipt.
11.          Client receive cash
12.         The use case ends.
identify self
verify identity
offer choice
choose
dispense cash
```

Figure 4. 4: Our Automated Tracing Tool

The tool processes are:

I. Textual natural language requirements are written in the textual authoring tool (1). The textual natural language requirements are expressed in natural language phrases. These may include headings, numbered items and bullet points as well as sentences. In this example, for clarity we used a numbered list of sentences. However, in general, the textual natural language requirements can contain other layouts (e.g. paragraphs) as appropriate. The requirement engineer can author this textual natural language requirement either in our authoring tool or in any external word processor, or can extract the text from an existing document such as Text File, PDF, Word, and Power Point files.

II. A list of corresponding essential requirements (abstract interactions) is generated automatically as shown in (2) using the button "Trace".

III. Users can trace back each abstract interaction to the corresponding textual requirements phrases as shown in (3) using the button "Trace Back".

IV. The engineer then asks the tool to extract all recognised EUC "essential interactions" expressed in the textual natural language requirements, using an essential interaction pattern library. The extracted essential interactions are shown in sequence as recognised in the text (2).

V. Depending on the complexity of the submitted requirements text, several EUC interaction sequences, or Essential Use Cases, may be recognised. These can be divided or represented as a collection of EUCs. We used a listing of these essential interaction phrases. These can be represented as an EUC model with user intention/system responsibility divisions using Constantine & Lockwood's [4] approach if desired.

VI. Users can interact with either the textual natural language requirement segments or the essential interactions extracted, in order to trace between the textual phrases and the essential interactions. Essentially, this provides a traceability mechanism between each abstract interaction to the corresponding textual natural language requirements phrases, as shown in the example of highlighting in (3). This tracing process helps requirements engineers to check for correctness, completeness and consistency of the requirements. Phrases with missing EUC essential interactions may be incorrect or incomplete. Phrases with too many corresponding essential interactions may be imprecise. A sequence of essential interactions with phrases in different parts of the textual requirements may mean the text requirements are out of order. A sequence of essential interactions that is incomplete or redundant may mean the textual natural language requirements have inconsistencies or undue repetition.

4.4 Evaluation

We carried out an evaluation of our automated tracing tool in order to compare its accuracy and performance with the manual extractions undertaken by our original EUC extraction study participants, described in Chapter 3. In addition, these same participants were asked to use and evaluate the automated tracing tool using the same scenario as before immediately after they had finished the manual study. They were also told about this evaluation before they started the first study. They were also told about this evaluation before they started the first study. Each participant was given a brief tutorial on how to use the tool and some examples of how the traceability provided by the tool is able to extract the abstract interaction from a set of textual natural language requirements and to identify its associated essential interaction. They then explored the traceability and extracted the abstract interaction for the scenario of "getting cash" which was illustrated in the previous chapter. We then surveyed them to gain their perceptions of the tool's usefulness and ease of use for the extraction and tracing tasks evaluated. We also asked for their open feedback on the tool's features and performance. The survey consisted of three questions for each question block of usefulness and ease of use. A five-part Likert scale was used for each question. For each characteristic, the results of each corresponding block were averaged to produce the results shown in Figures 4.5 and 4.6. The type of questions for each characteristic is in Table 4.3. The participants' open feedback is shown in Table 4.4.

User Perception Characteristics	Questions
Usefulness	The tool is useful in finding the abstract interaction.
	The tool helps me to be more effective in extracting the textual natural language requirement to the Abstract interaction.
	It is easier to capture the core requirement by using the automated tracing tool compared to the manual extraction
Ease of Use	It is easy to use.
	It is user friendly.
	I don't notice any inconsistencies as I use the tool.

Table 4. 3: User Perception Characteristics and Questions Evaluating Them

Participants	Feedback
1.	" The tool is easy to use but not interesting."
2.	"The tool can enhance the ability by selecting all abstract interactions and then trace back."
3.	"The tool is easy but I think it will have constraints with the database."
4.	"Limited coverage of phrases"
5.	"Hope tool can have more interactive visual; for example colours and shapes"
6	"OK."
7.	"Easy to use and easy to understand."
8.	" More data set for library"
9.	"It supports any size of files and brings fast results."
10.	None
11.	"The library needs enhancement as it does not support certain phrases and words"

Table 4. 4: Participants Open Feedback

The results of the participant survey of the tool usefulness and ease of use are shown in Figures 4.5 and 4.6 respectively. All eleven participants found that the tool was either very useful (85%) or always useful (15%) for generating and tracing the list of abstract interaction. However, in qualitative feedback, most participants wanted the interaction pattern library to support a broader set of domains in the future.

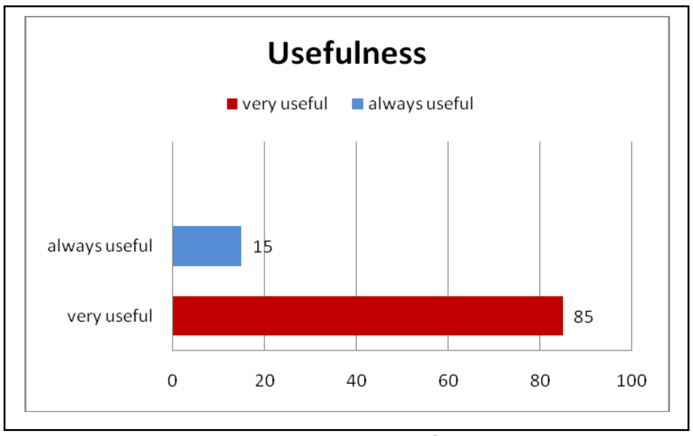


Figure 4. 5: The Tool Usefulness Results

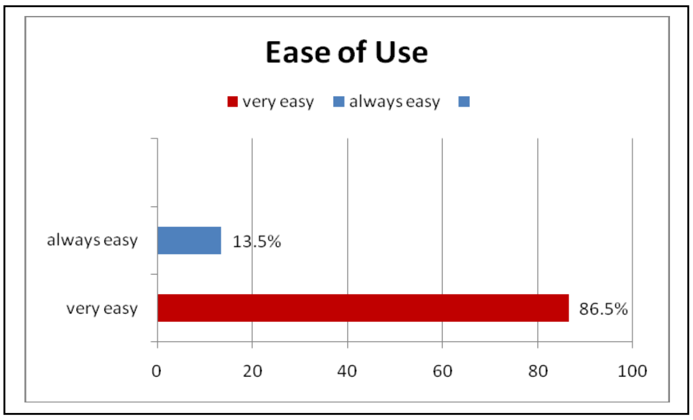


Figure 4. 6: The tool Ease of Use Results

The results in Table 4.5 compare the accuracy of the automated tracing tool against the previous results for manual extraction. The tool succeeded in identifying almost all the abstract interactions, failing to detect one abstract interaction, providing an accuracy of almost double the participants' average and better than all but one of the participants' accuracy. The correctness ratio for manual extraction is only 47% and the automated tracing tool provides 83%. The single error from the tool is because of its failure to detect one of the abstract interactions (Take Cash). The automated extraction process took just over one second to execute in comparison with the 11.2 minutes average taken by the manual study participants.

Answers	No. Correct answers		No. Wrong answers	
	Manual extraction	Automated Tracing	Manual extraction	Automated Tracing
Identify user	5	1	6	0
Verify Identity	4	1	7	0
Offer cash	4	1	7	0
Choose	6	1	5	0
Dispense cash	9	1	2	0
Take cash	3	0	8	1
Correctness ratio	47%	83%	53%	17%

Table 4. 5: Comparison result of correctness between Manual extraction (previous chapter) and Automated Tracing Tool

To further investigate the utility of our tool, we evaluated its accuracy when applied to 15 use case scenarios in different domains derived from different researchers, developers and ourselves: Online CD catalogue, Cellular phone [23], Voter registration [125] Cash withdrawal [126], Online book [127], Checkout book (library) [2], Seminar Enrolment [4], Transfer transaction [126], Deposit transaction [126], Assign report problem [128], Create problem report [128], Report problem [128], Booking room [129] and Place order [130]. The tool correctness was evaluated by comparing the answers with the actual interaction pattern provided in the source pattern documents that was developed by Constantine and Lockwood [4], Biddle et al. [39] and also with patterns developed by us following Constantine and Lockwood's methodology. The correct or similar results provided by the tool are calculated and averaged with the actual interaction pattern provided for a particular scenario. The comparison result is then valued with a ratio of maximum 1. The evaluation results are shown in Figure 4.7.

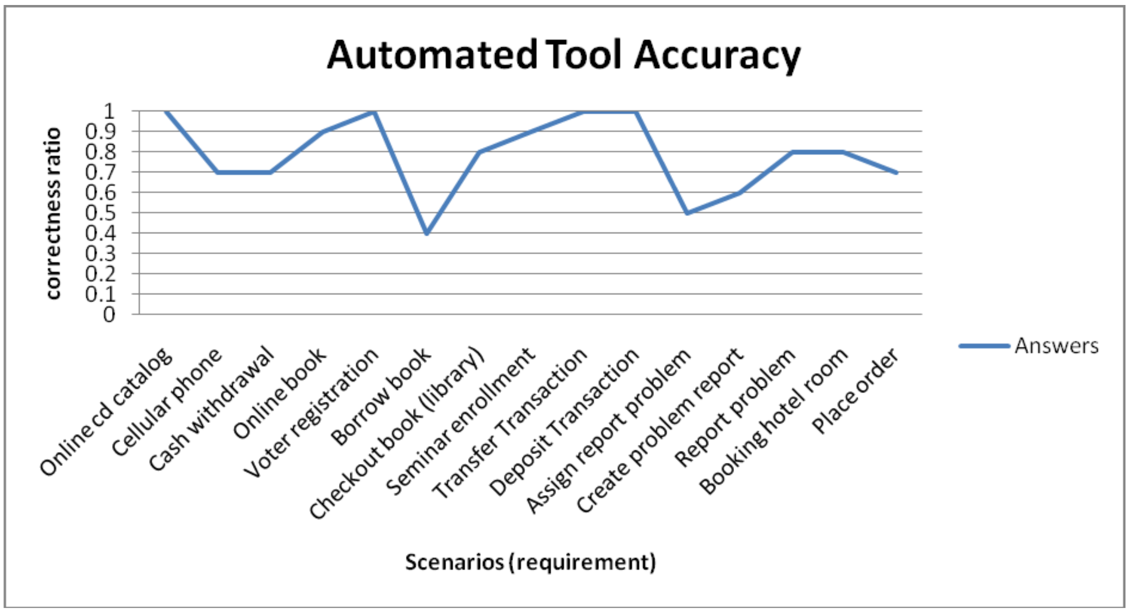


Figure 4. 7: Accuracy across different scenarios

Figure 4.7 shows the correctness ratio for the automated tracing tool for each scenario. This shows some variability across the range of scenarios, but the average correctness across all scenarios and interactions is approximately 80%, so the “getting cash” scenario used in the earlier evaluation was not atypical. The automated tracing tool does not (and cannot) produce 100% correct answers due to the incorrectness and incompleteness issue of textual requirements. The correctness and incompleteness issue is related to various linguistic issues, such as phrases or sentences using a passive pattern, parentheses’ existence such as {,,[,],/,\ and grammar issues such as plural, singular, adjective or adverb issues[134]. These problems, however, also lead requirement engineers to misunderstand requirements and can be one of the reasons why different requirement engineers or users provide inconsistent results.

For example, our automated tracing tool did not derive a completely correct EUC interaction for the scenario “Getting Cash” because the grammar used in the sentences of the textual natural language requirements was incorrect. The phrase “receive cash” from sentence number 11: “Client receive cash” is not readable by the tool as in the database, it is stored as “receives cash”. This problem can be improved either by giving guidelines to users for writing a good requirements document or allowing the library to be expanded to accept grammatically incorrect sentences for patterns that correspond to common grammatical errors. Additionally, we have experimented with using simple regular expressions in the essential interaction pattern repository e.g. “receive{s} cash”, indicating it should have an ‘s’ but may accept without. This, however, complicates both the library phrase representation and the authoring.

Using our tool, requirements engineers will notice that the “receive cash” phrase in the textual requirements does not have any corresponding essential interaction phrase(s). Alternatively, they will see an incomplete interaction sequence between the user and the system where no response is provided to a user submission by the system in the EUC model extracted and visualised. In our Eclipse-based prototype described in chapter 5, we have experimented with adding checking for such apparent inconsistencies between requirement texts and essential interactions. This is also complicated by textual natural language requirements, typically having portions of texts that do not correspond directly to the interactions such as headings, introductory or concluding remarks, comments, and example input/output data.

Overall, our automated tracing tool still has several limitations. Firstly, it is stand-alone and does not integrate with other requirement or software engineering tools. This causes the tool to appear to have fewer benefits to the user. Next, it also has limited visual interface or aspects which lead the user to not understand well the process and the usage of the tool. In addition, the tool has constraints with the database of the interaction pattern library. The database needs enhancement with more phrases from a broader set of domains.

4.5 Summary

We have discussed our approach in extracting essential interactions using a more lightweight approach. We have developed an automated prototype EUC essential interaction extraction and tracing tool. The key aims of our tool were to support EUC by extracting the essential requirements (abstract interactions) automatically and to facilitate tracing between EUC and textual natural language requirements to assist engineers in identifying and managing inconsistencies and incompleteness. Another aspect of our research involved collecting and categorising terminology for the library of abstract interactions. This both assists in structuring EUC expressed requirements using common terminology and also helps prevent the textual natural language requirements from being vague and error-prone, by tracing back from the EUC-structured representations to the textual natural language requirement phrases.

We have evaluated our prototype tool using the same group of participants that we used for the manual extraction survey described in the previous chapter. The participants evaluated the tool’s usefulness and ease of use with promising results. This confirms other researchers’ claims about the importance of having tool support for engineers working with EUC models. Our results found that such automated extraction and the tracing tool appear to increase the ratio of correctness in extracting EUC requirements from textual natural language requirements and ease the effort of users or requirement engineers in handling the EUC, significantly reducing the time taken to develop EUC models from textual natural language requirements.

This is the first phase of our incremental work. The next phase, described in the following chapter, focuses on embedding our extraction approach into an integrated EUC Diagram tool (Marama Essential) which was developed using the Marama meta tool [120], to overcome the problems faced in our current tool, and managing the consistency of requirements by adding more support for inconsistency detection using our extraction approach and round-trip engineering of natural language and EUC model requirements.

Chapter 5: Managing Requirements Consistency

This chapter describes our approach to improving the consistency management of requirements by embedding our automated tracing tool, as discussed in the previous chapter, in the Marama Meta tool [120] together with additional support for inconsistency detection using our extraction approach and round-trip engineering of natural language and EUC model requirements. To enhance consistency management, traceability and visualisation capability are applied.

5.1 Introduction

Consistency management between different artefacts in software engineering has been recognised as vital for many years [27],[33],[131],[132]. Consistency management between formal requirement specifications and architecture and design models has been investigated, especially in the Requirement Engineering domain [29],[26]. Similarly, several approaches have been developed to try to determine inconsistencies between natural language descriptions of requirements and formalised models of requirements [27],[34]. Some techniques have been developed to support the correction of inconsistencies such as the use of repair operations [76]. Detecting inconsistencies may or may not require immediate correction. Living with inconsistency requires the management of inconsistencies over time: this provides more flexibility in the development process [22]. Correcting inconsistencies and providing appropriate tool support to detect, present and manage these inconsistencies are identified as being challenging [133].

5.2 Managing the consistency

We have devised another approach that is applied together with a traceability technique to help support consistency management between textual natural language requirements and the EUC model. This work focuses on managing the essential interaction requirements to capture the functional requirements of a system. We have created an “essential interaction” phrase library from the collection and the categorisation of requirements from different domains and scenarios as described in Chapter 4. Phrases have been extracted and stored in this library and are used to match corresponding phrases in textual natural language requirements. The extracted phrases are further mapped to specific abstract interactions. Each abstract interaction is classified as a user intention or a system responsibility. The derived essential use case elements can be traced back to their originating natural language requirements phrases and vice-versa. We embed this extraction and tracing support into an Essential Use Case editing tool that we have developed using the Marama meta-tool platform [120] . This provides an environment in which requirements engineers have the ability to extract and then generate candidate diagrammatic EUCs automatically from requirements expressed in textual natural language. Consistency management support is then provided between these textually-expressed requirements, a derived set of structured abstract interaction and semi-formal diagrammatic EUCs. Requirements engineers can move between different requirement forms, using the traceability relationships preserved during the extraction and generation processes. They can modify any one of the requirement forms from the informal textual natural language to the semi-formal EUC diagrams. The environment will attempt to update the other forms and/or indicate resultant inconsistencies.

The framework for extracting the requirements, mapping the types of interaction and creating the EUC model is shown in Figure 5.1**.** This illustrates the extraction of a set of abstract interactions from the textual natural language requirements. The library of abstract interaction phrases is used by a “trace engine” to analyse the text for matches and a set of candidate abstract interactions generated (1). A “mapping engine” then uses a database of essential interaction to structure the interactions into an EUC model (2). The mapping engine then generates a diagrammatic representation of the Essential Use Case (3) which represents the dialogue occurring between the user and the system. The traceability relationships among elements in the textual natural language requirements model, the extracted essential interactions model and the diagrammatic EUC model are preserved and can be used to support traceability between the three forms of requirements and to check for inconsistencies among them. Examples of inconsistencies are elements identified in one form but not in another, inconsistent naming, the ordering or properties of elements and duplicated or partially duplicated phrases or elements.

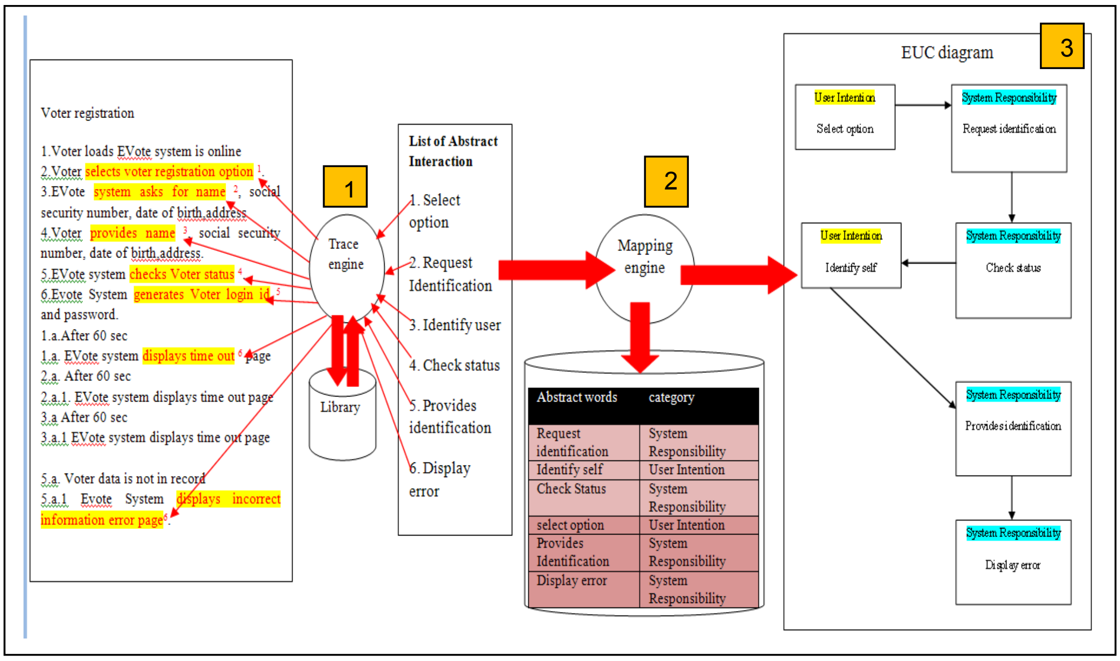
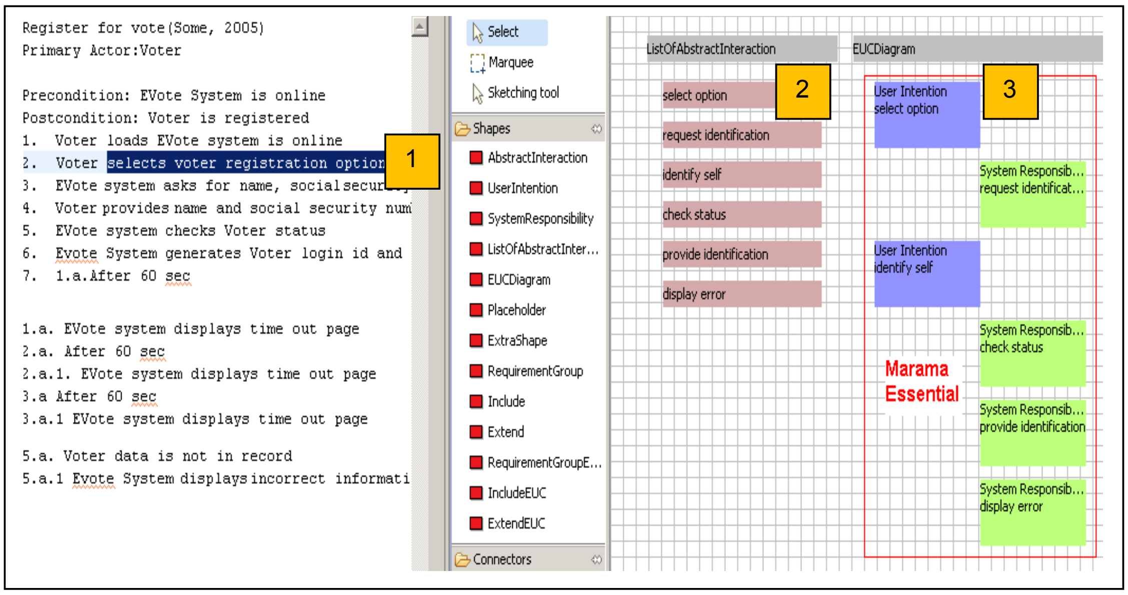

Voter registration
1.Voter loads EVote system is online
2.Voter selects voter registration option 1.
3.EVote system asks for name 2, social security number, date of birth,address.
4.Voter provides name 3, social security number, date of birth,address.
5.EVote system checks Voter status 4
6.Evote System generates Voter login id 5 and password.
1.a.After 60 sec
1.a. EVote system displays time out 6 page
2.a. After 60 sec
2.a.1. EVote system displays time out page
3.a After 60 sec
3.a.1 EVote system displays time out page
5.a. Voter data is not in record
5.a.1 Evote System displays incorrect information error page6.
1
Trace engine
Library
List of Abstract Interaction
1. Select option
2. Request Identification
3. Identify user
4. Check status
5. Provides identification
6. Display error
2
Mapping engine
Abstract words
category
Request identification
System Responsibility
Identify self
User Intention
Check Status
System Responsibility
select option
User Intention
Provides Identification
System Responsibility
Display error
System Responsibility
EUC diagram
3
User Intention
Select option
System Responsibility
Request identification
User Intention
Identify self
System Responsibility
Check status
System Responsibility
Provides identification
System Responsibility
Display error

Figure 5. 1: Framework for Extracting Requirement (1) Mapping interactions (2) and Creating the EUC Automatically (3)

5.3 Tool Support

Based on the framework in Figure 5.1, we have developed an Automated Inconsistency Checker, MaramaAI, together with an EUC diagram editor, Marama Essential. This work provides support to EUC users and requirements engineers for capturing requirements, designing and generating EUCs automatically and minimising the time to develop them from source textual natural language requirements. This increases the correctness of the abstract interactions produced. In addition, this automated tool helps to lessen the need for human intervention in capturing requirements and checking the consistency of software requirements. The tool provides consistency checking and notification support, allowing requirements engineers to modify any of the three forms of requirements in the tool. Any new abstract interaction or EUC component can be inserted and updated. These changes will trigger the tool to automatically perform consistency checking. In addition, the textual natural language requirements can also be added, modified or deleted. For this, the tool will only perform consistency checking after the requirements engineer invokes the event handler. The event handlers are explained in detail in a later section. We used the Marama meta-toolset [120], a set of Eclipse IDE plug-ins, to develop our MaramaAI prototype. MaramaAI allows traceability to be interactively visualised in the textual natural language requirements, abstract interaction and EUCs. Further, any requirements that are inconsistent and incomplete can be highlighted to the user and a warning is provided. The tool also comprises a glossary and set of guidelines adapted from [134] to assist users to write correct and complete EUC-based requirements. We do not use all the guidelines as they contain many conditions which might constrain the description of requirements. The implementation of all guidelines is also quite difficult for this proof of concept phase and so we have deferred full implementation to future work.We only adopt the common guidelines that all sentences must be in active not passive voice , and that the written requirements should avoid the use of brackets or parentheses [134]. It is believed that parentheses and brackets can lead to ambiguity and interpretation problems for the contents of the requirements as they may lead to uncertain numbers of requirements in the sentences [134]. This is also to overcome the problems faced by our previous automated tracing tool in handling grammar and the parentheses issue revealed by the study outlined in Chapter 4. That study concerned the accuracy in terms of correctness of the abstract interaction provided by our automated extraction feature, using the interaction patterns provided by the library based on the collection of patterns from Constantine and Lockwood [4], Biddle et al. [39], and patterns developed by us. The results show that the incorrectness and incompleteness of textual natural language requirements seriously affect the ability to produce correct abstract interactions in order to structure the requirements.

Our automated extraction and tracing tool which we embed in the Marama meta-tool is shown in Figure 5.2. For this, we consider the scenario of a “voter registration” use case by Some [125] using an illustration. We use this scenario as a case study to show the benefits and the flow of the consistency checking process. Figure 5.2 shows a set of requirements for this voter registration system that is expressed in natural language and opened in an Eclipse text editor (1). The textual natural language requirements do not have to be structured as a list nor formed into a structured layout as shown in this example. The requirements engineer has used the tool to analyse these requirements and a set of “abstract interactions” has been deduced from these textual natural language requirements. These abstract interactions are then represented as a vertical list (2). Our tracing engine uses an essential interaction library of phrases and regular expressions to deduce and extract candidate abstract interaction from the associated essential interactions. From the abstract interaction list thus extracted, our mapping engine generates an EUC diagram (3) using a set of abstract interaction patterns and EUC diagram heuristics as shown below with a red box. The user can interact with the three representations of requirements- the natural language expressed as textual natural language requirements, the abstract interactions which is the essential requirement, and the diagrammatic EUC model in Marama Essential.

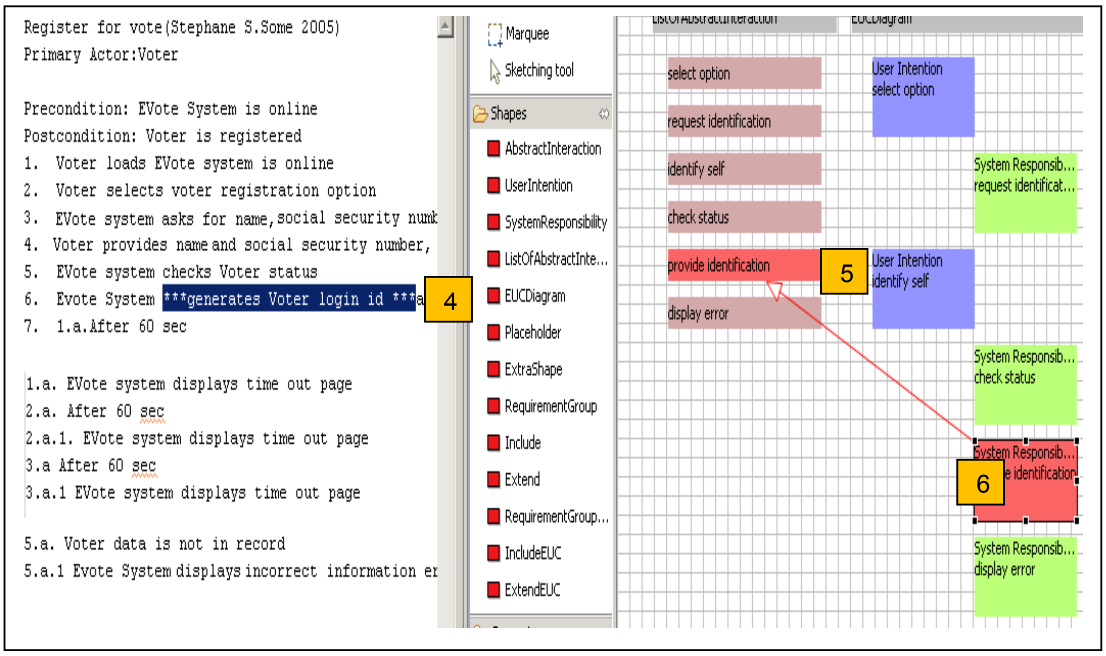


```
Register for vote(Some, 2005)
Primary Actor:Voter

Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter selects voter registration option
3.  EVote system asks for name, social secur
4.  Voter provides name and social security num
5.  EVote system checks Voter status
6.  Evote System generates Voter login id and
7.  1.a.After 60 sec

1.a. EVote system displays time out page
2.a. After 60 sec
2.a.1. EVote system displays time out page
3.a After 60 sec
3.a.1 EVote system displays time out page

5.a. Voter data is not in record
5.a.1 Evote System displays incorrect informati
```

Figure 5. 2: Tracing the Abstract Interaction from Textual Natural Language Requirement and Mapping to the Marama Essential

For example, as shown in Figure 5.2 (1), the selected phrase – “select voter registration option” is traced to a particular abstract interaction – “select option” (2). This has then been mapped to the EUC diagram and falls under the “user intention” category (3) and select option interaction.
This shows that the tool provides a traceability link for the three requirement components above. The tool not only provides trace-forward but can also is able to trace-back from an EUC diagram to abstract interactions and then to a textual natural language requirements. The process of tracing back is shown in Figure 5.3.

In Figure 5.3, the item “provide identification” (6) from the system responsibility category of the EUC diagram is selected. This highlights the selected EUC component and the related essential requirement, which in this case is “provide identification” (5). The traceability between these items is shown by the visual link (red arrow). The corresponding textual natural language phrases are automatically highlighted, the matched abstract interaction changes colour to red in (4) and the matched phrases are quoted with *** (6). The existence of these traceability links allows consistency among these three items to be maintained.



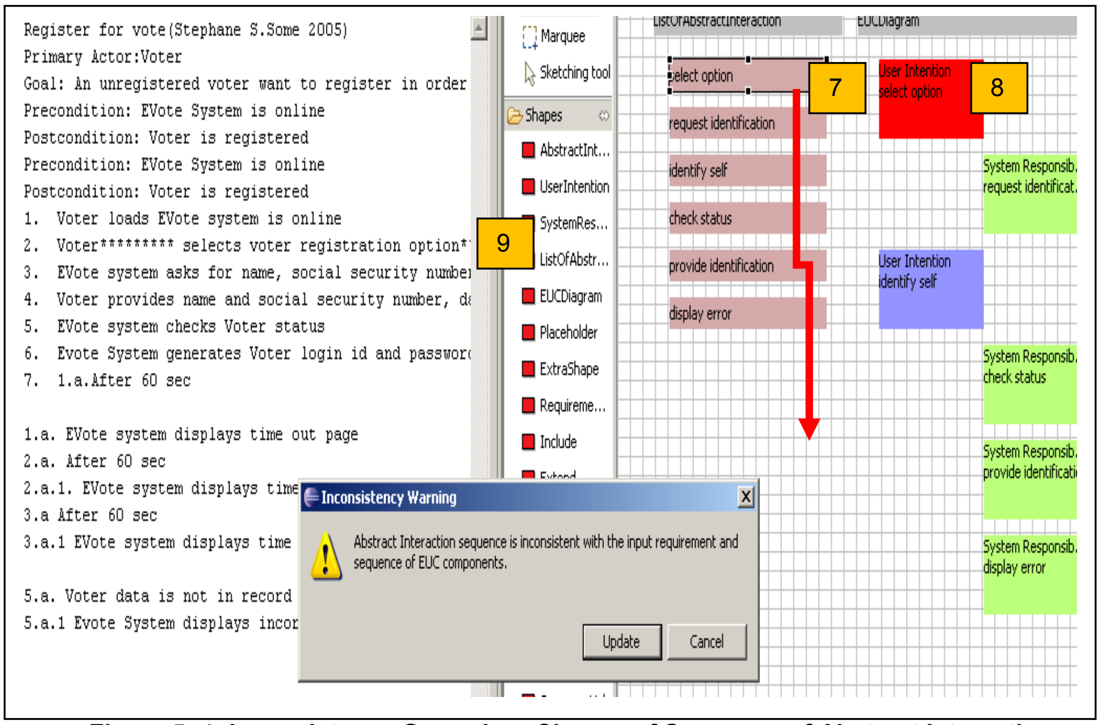
```
Register for vote(Stephane S.Some 2005)
Primary Actor:Voter

Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter selects voter registration option
3.  EVote system asks for name,social security num
4.  Voter provides name and social security number,
5.  EVote system checks Voter status
6.  Evote System ***generates Voter login id ***a
7.  1.a.After 60 sec

1.a. EVote system displays time out page
2.a. After 60 sec
2.a.1. EVote system displays time out page
3.a After 60 sec
3.a.1 EVote system displays time out page

5.a. Voter data is not in record
5.a.1 Evote System displays incorrect information er
```

Figure 5. 3: Trace back from EUC diagram in Marama Essential to the Abstract Interaction and Textual natural language requirement

As shown in Figure 5.3, the traceability link provided is also a possible way to inform the requirements engineer if any item appears to be incomplete or incorrect by identifying the links for each of the requirements components. The requirements engineer may modify any one of these requirement components and the tool will check the resulting models, both for the internal model consistency, which involves only the Essential Use Case in the Marama Essential view, and the inter-model consistency which involves all three views: textual natural language requirement, abstract interaction and the EUC diagram.

Inconsistency occurs if any change or modification is made to the components [76]. Thus, this view is applicable to our work. If any inconsistency occurs due to a change made by the user, for example if there is a change of order, name or type for any of the abstract interaction or EUC diagram elements, an inconsistency warning will occur. If an item or phrase has been added and the new item cannot be matched to a textual natural language requirement phrase or abstract interaction by the tracing engine, an inconsistency warning will also occur. If traceability relationships do not exist between phrases and items, this indicates the existence of a potential incompleteness or inconsistency, which means that no tracing result will be shown to the engineer. The tool can highlight items for the engineer to investigate in one view that do not appear to be related to items in another.

Various types of inconsistency errors described are shown in Figures 5.4, 5.5, 5.6 and 5.7. In Figure 5.4, item (7) shows an example of inconsistency error when a change of sequence to an abstract interaction - “select option” is made. Our tool provides the flexibility where requirements engineers can verify the abstract interaction provided, before mapping it to the EUC diagram. From the example, the requirements engineer did not agree with the position set by the tool and decided this should be in a different position in the set of abstract interactions. The tool has highlighted the potential inconsistency of the EUC diagram in red (8), and the associated phrase “select voter registration option” from the textual natural language requirements (9) is highlighted with (***). This change also leads to a change of sequence and position in the EUC diagram - “select option” (8) - to the bottom, which is highlighted in red. The red arrows show the change of sequence from the original position to the new one. This produces an inconsistency in the requirements as the textual natural language requirement remains unaltered. This is because there is no automated update for the structure of the sentences as it is believed that might affect the whole structure of the sentences. The tool however does detect the inconsistencies and provides a warning about them.

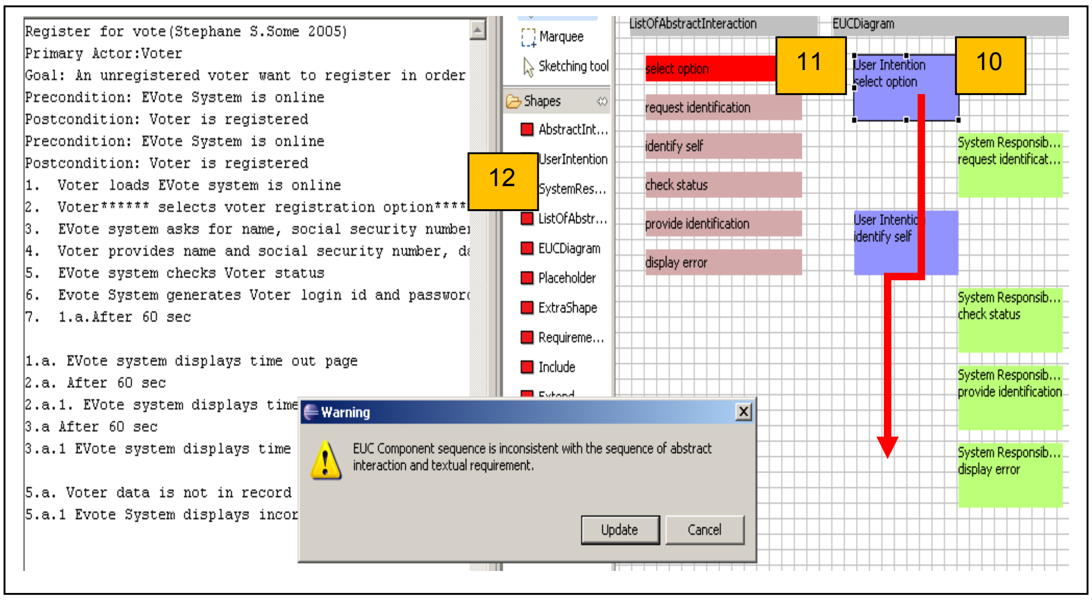


```
Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to register in order
Precondition: EVote System is online
Postcondition: Voter is registered
Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter********* selects voter registration option*
3.  EVote system asks for name, social security numbe
4.  Voter provides name and social security number, d
5.  EVote system checks Voter status
6.  Evote System generates Voter login id and passwor
7.  1.a.After 60 sec

1.a. EVote system displays time out page
2.a. After 60 sec
2.a.1. EVote system displays time
3.a After 60 sec
3.a.1 EVote system displays time

5.a. Voter data is not in record
5.a.1 Evote System displays incor
```

Inconsistency Warning

Abstract Interaction sequence is inconsistent with the input requirement and sequence of EUC components.

Figure 5. 4: Inconsistency Occurring: Change of Sequence of Abstract Interaction

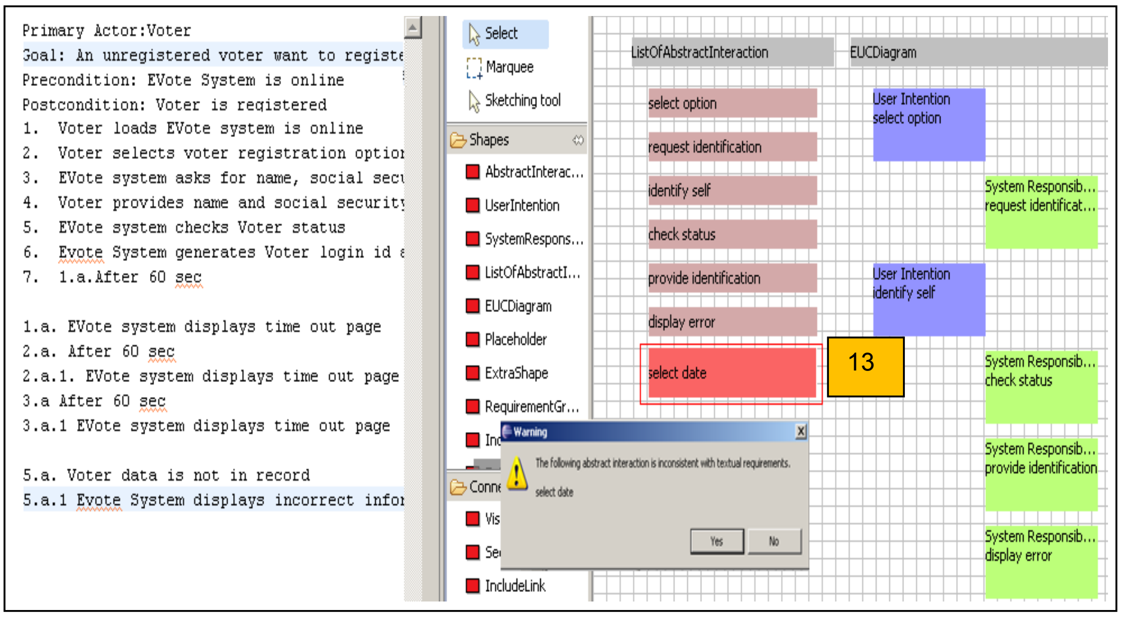

```
Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to register in order
Precondition: EVote System is online
Postcondition: Voter is registered
Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter****** selects voter registration option****
3.  EVote system asks for name, social security numbe
4.  Voter provides name and social security number, d
5.  EVote system checks Voter status
6.  Evote System generates Voter login id and passwor
7.  1.a.After 60 sec

1.a. EVote system displays time out page
2.a. After 60 sec
2.a.1. EVote system displays time
3.a After 60 sec
3.a.1 EVote system displays time

5.a. Voter data is not in record
5.a.1 Evote System displays incor
```

Warning

EUC Component sequence is inconsistent with the sequence of abstract interaction and textual requirement.

Figure 5. 5: Inconsistency Occurring: Change of Sequence of EUC component

Figure 5.5 shows the inconsistency occurring when any EUC component is changed to a new position different from the auto generated position. In this example, an EUC component “select option” (10) from the user intention category is moved to the bottom, as shown by the red arrow. The corresponding abstract interaction “select option” (11) is highlighted in red and the associated phrases “select voter registration option” from the textual natural language requirements (12) is highlighted with (***). The red arrows show the change of sequence from the original position to the new one. This produces an inconsistency in the requirements as the textual natural language requirement remains unaltered.



```
Primary Actor:Voter
Goal: An unregistered voter want to registe
Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter selects voter registration optior
3.  EVote system asks for name, social secu
4.  Voter provides name and social security
5.  EVote system checks Voter status
6.  Evote System generates Voter login id a
7.  1.a.After 60 sec

1.a. EVote system displays time out page
2.a. After 60 sec
2.a.1. EVote system displays time out page
3.a After 60 sec
3.a.1 EVote system displays time out page

5.a. Voter data is not in record
5.a.1 Evote System displays incorrect infor
```

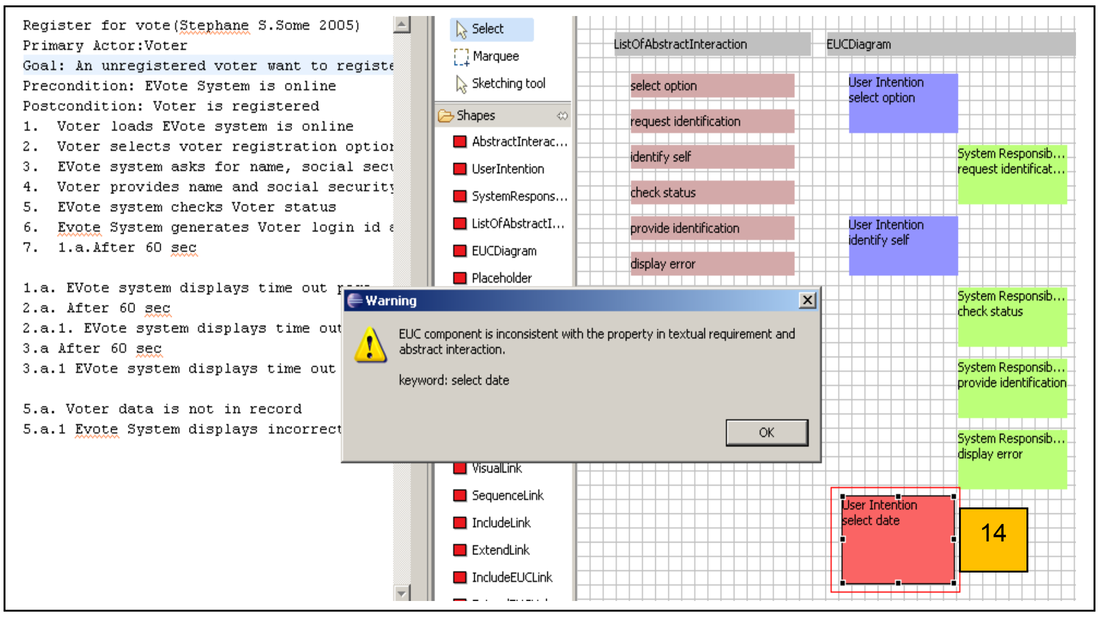
Warning

The following abstract interaction is inconsistent with textual requirements.

select date

Figure 5. 6: Inconsistency Occurring: Adding New Item to the Abstract Interaction

Figure 5.6 shows a potential inconsistency that happens when a new item is added to the abstract interaction. It shows a new abstract interaction, “select date”, has been inserted into the abstract interaction view (13). When the tool checks the new abstract interaction with the textual natural language requirements, it detects an inconsistency between the new abstract interaction “select date” and the textual natural language requirements. This triggers an inconsistency warning to appear and highlights the new added item” select date” in red. The warning informs the requirements engineer where the inconsistency is located.


```
Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to registe
Precondition: EVote System is online
Postcondition: Voter is registered
1.   Voter loads EVote system is online
2.   Voter selects voter registration optior
3.   EVote system asks for name, social secu
4.   Voter provides name and social security
5.   EVote system checks Voter status
6.   Evote System generates Voter login id a
7.   1.a.After 60 sec

1.a. EVote system displays time out p
2.a. After 60 sec
2.a.1. EVote system displays time out
3.a After 60 sec
3.a.1 EVote system displays time out

5.a. Voter data is not in record
5.a.1 Evote System displays incorrect
```

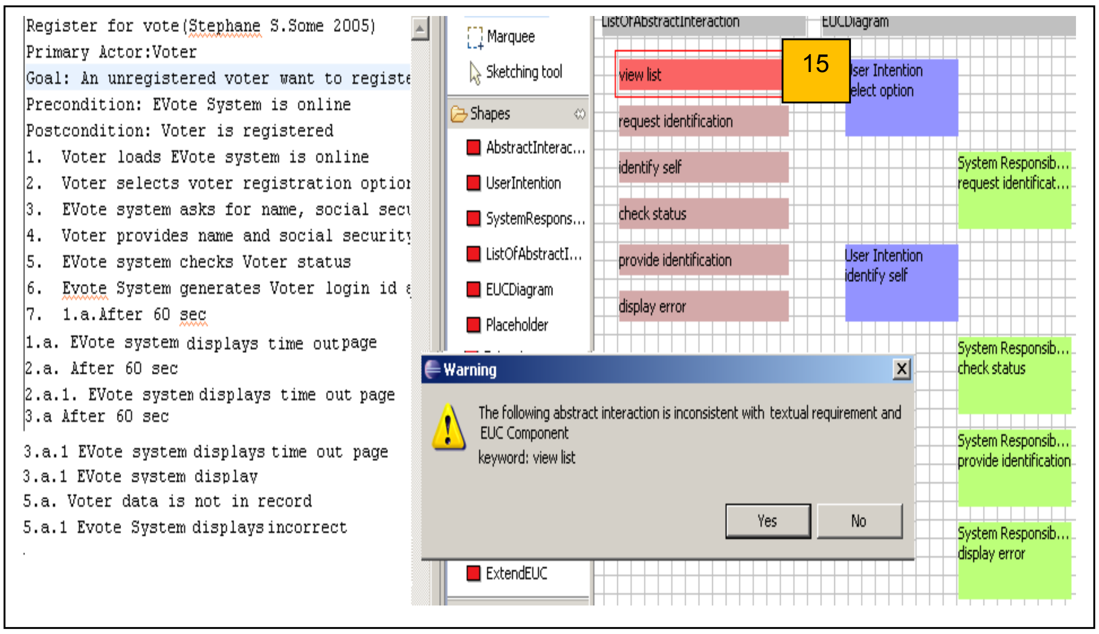
Warning

EUC component is inconsistent with the property in textual requirement and abstract interaction.

keyword: select date

Figure 5. 7: Inconsistency Occurring: Adding New Item to the EUC diagram

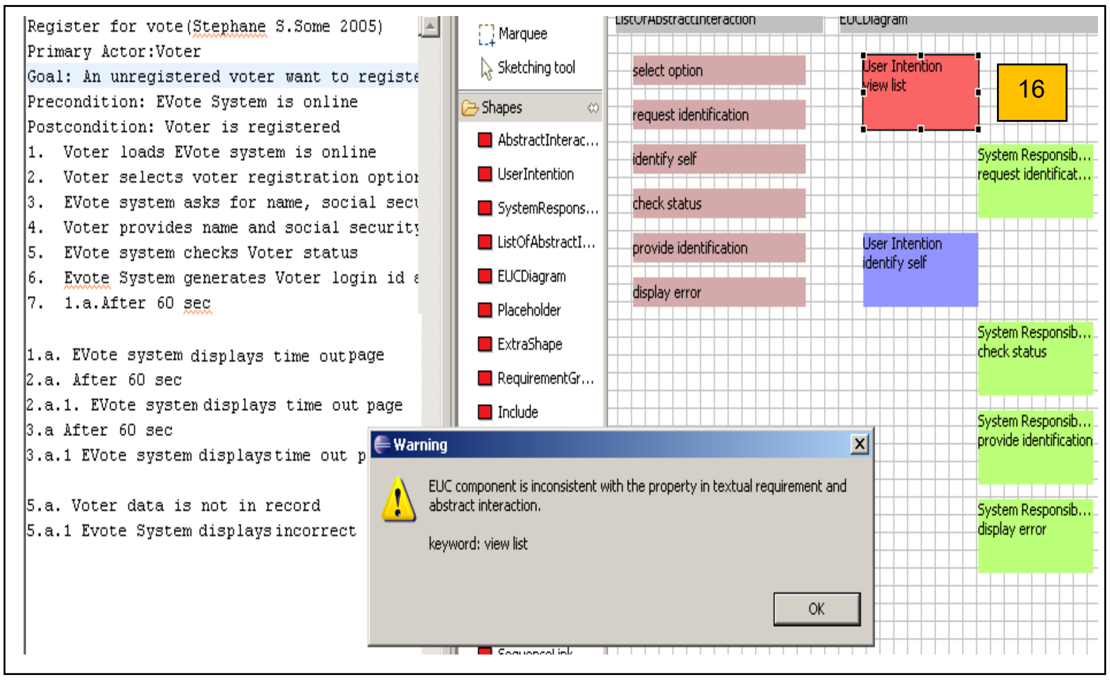
Figure 5.7 shows a potential inconsistency that happens when a new item is added to the EUC diagram in the Marama Essential. It shows that a new EUC component, “select date”, has been inserted into the EUC diagram view (14). When the tool checks the new EUC component with the abstract interaction and textual natural language requirements, it detects inconsistencies between the new EUC component “select date” and the abstract interaction and the textual natural language requirements. This triggers an inconsistency warning to appear and highlights the new, added item ” select date” in red. The warning informs the requirements engineer where the inconsistency is located.


```
Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to registe
Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter selects voter registration optio
3.  EVote system asks for name, social secu
4.  Voter provides name and social security
5.  EVote system checks Voter status
6.  Evote System generates Voter login id 
7.  1.a.After 60 sec
1.a. EVote system displays time outpage
2.a. After 60 sec
2.a.1. EVote systen displays time out page
3.a After 60 sec

3.a.1 EVote system displays time out page
3.a.1 EVote system display
5.a. Voter data is not in record
5.a.1 Evote System displays incorrect
```

The following abstract interaction is inconsistent with textual requirement and EUC Component
keyword: view list

Figure 5. 8: Inconsistency Occurring: Change of Name to the Abstract Interaction

Figure 5.8 shows the inconsistency occurring when the name for an abstract interaction is changed. As shown, the abstract interaction component “select option” is changed to “view list” (15). When the tool checks the changed component “view list” with the textual natural language requirement and EUC diagram, it detects inconsistencies between the changed abstract interaction component “view list” and the textual natural language requirements and the EUC component. This triggers an inconsistency warning to appear and highlights the changed item ”view list” in red. The warning informs the requirements engineer where the inconsistency is located.



```
Register for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to registe
Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter selects voter registration optio
3.  EVote system asks for name, social secu
4.  Voter provides name and social security
5.  EVote system checks Voter status
6.  Evote System generates Voter login id a
7.  1.a.After 60 sec

1.a. EVote system displays time outpage
2.a. After 60 sec
2.a.1. EVote systen displays time out page
3.a After 60 sec
3.a.1 EVote system displaystime out p

5.a. Voter data is not in record
5.a.1 Evote System displays incorrect
```

Warning

EUC component is inconsistent with the property in textual requirement and abstract interaction.

keyword: view list

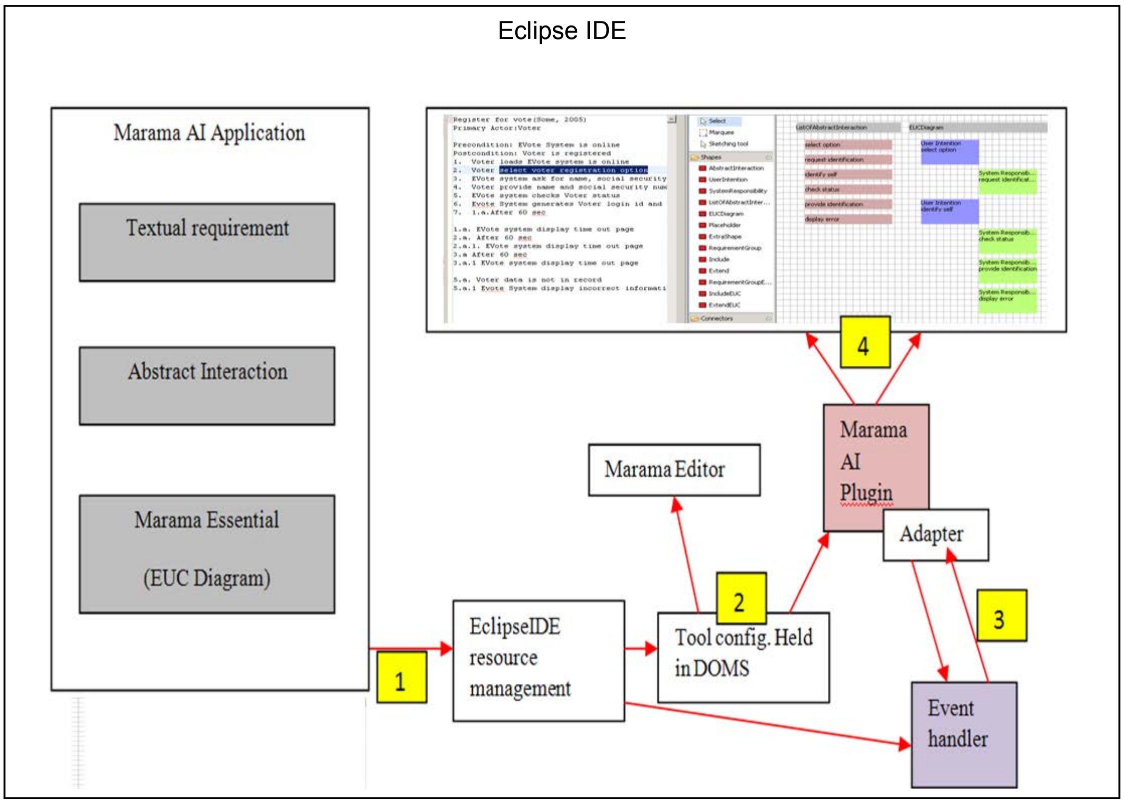
Figure 5. 9: Inconsistency Occurring: Change of Name to the EUC component

Figure 5.9 shows the inconsistency occurring when the name of an EUC component is changed. The EUC component “select option” is changed to “view list” (16). When the tool checks the changed component “view list” with the abstract interaction and the textual natural language requirements, it detects inconsistencies between the changed EUC component “view list” and the abstract interaction and the textual natural language requirements. This triggers an inconsistency warning to appear and highlights the changed item ”view list” in red. The warning informs the requirements engineer where the inconsistency is located.

To sum up, these inconsistency warnings and highlights shown in the figures above illustrate the dependencies that occur among the requirement components: the textual natural language requirements, abstract interaction and the EUC diagram. If any changes are made to any of the requirement components, the tool will check the change with the associated components. Any inconsistency detected will trigger an inconsistency warning to appear and also highlights the inconsistencies error in red for either abstract interaction or the EUC diagram and (***) for the textual natural language requirements.

5.4 Architecture and Implementation

As mentioned earlier, to capture and check the inconsistency, we embed our previous automated prototype tool in the Marama meta-tool [120]. Our new embedded tool is called MaramaAI. MaramaAI consists of a textual natural language requirement, abstract interaction and Marama Essential (EUC diagram) editors. The architecture of Marama AI is shown in Figure 5.10. MaramaAI is realised on Marama which is built in the Java–Eclipse platform (1-2). MaramaAI editors are specified using Marama shape, meta-model and view tools. The tool is then implemented by interpreting the specification using a set of Marama plug-ins (4). The process of extracting and mapping any of the requirement components is assisted by the event handlers (3). Here, the event handler is the vital agent in maintaining the consistency among the three forms of requirements listed earlier.



```
Register for vote(Some, 2005)
Primary Actor:Voter

Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter select voter registration option
3.  EVote system ask for name, social security
4.  Voter provide name and social security num
5.  EVote system checks Voter status
6.  Evote System generates Voter login id and
7.  1.a.After 60 sec

1.a. EVote system display time out page
2.a. After 60 sec
2.a.1. EVote system display time out page
3.a After 60 sec
3.a.1 EVote system display time out page

5.a. Voter data is not in record
5.a.1 Evote System display incorrect informati
```

Figure 5. 10: MaramaAI Architecture

Four types of event handlers are used to support the automation process of capturing the requirements and checking the inconsistencies. These are: Trace, Trace Back, Map Abstract interactions to EUC and Index Checker. The description of the event handlers is as follows.

I. Trace

The event handler for tracing the textual requirement to the abstract interaction is called Trace. Here, the tracing engine will extract the key phrases which will be analysed by the essential interaction pattern library to match the keyword (abstract interaction). If the key phrases match the keywords, the abstract interaction will be displayed. If there is no match between the key phrases and the keywords, no results will be displayed. This normally happens when the textual natural language requirements is incorrect or incomplete. A sequence chart to illustrate the interaction is shown in Figure 5.11 below.

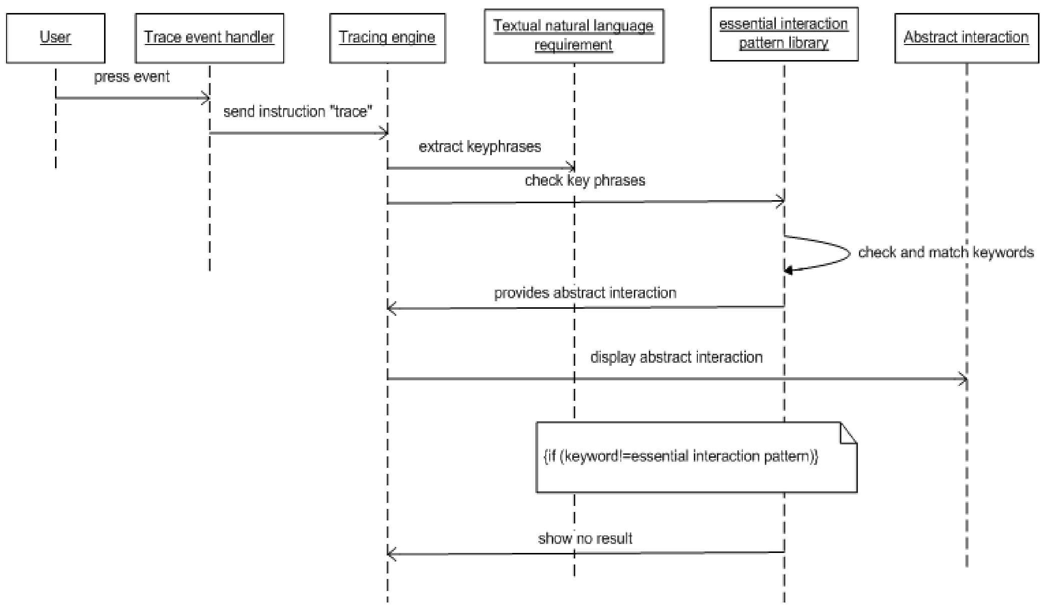

User
Trace event handler
Tracing engine
Textual natural language requirement
essential interaction pattern library
Abstract interaction
press event
send instruction "trace"
extract keyphrases
check key phrases
check and match keywords
provides abstract interaction
display abstract interaction
{if (keyword!=essential interaction pattern)}
show no result

Figure 5. 11: Example of Trace interaction

II. Trace back

To trace back the abstract interaction or EUC component from where it comes from, we use the help of the Trace Back event handler. This event handler also works together with the tracing engine. The selected abstract interaction or EUC component is analysed by the tracing engine and then matched with key phrases in the interaction pattern library. If we try to trace back the abstract interaction, the tool will show where the key phrases for that particular abstract interaction come from. If we trace back the EUC component, the system will show which abstract interaction matches it, together with the matching key phrases in the textual requirement. If no result appears, it is presumed that the requirements is either incorrect or incomplete. The requirement is also inconsistent if users try to change the requirement by adding new abstract interactions or EUC components or change the name of any of these components as shown in Figures 5.6, 5.7, 5.8 and 5.9. The trace back event handler will not be able to trace the key phrases in the textual natural language requirements as the new component is added or changed without updating the textual natural language requirement. This will also trigger the inconsistency warning to occur. To show further the interaction process of this event handler, a sequence chart to illustrate the interaction is shown in Figures 5.12 and 5.13.

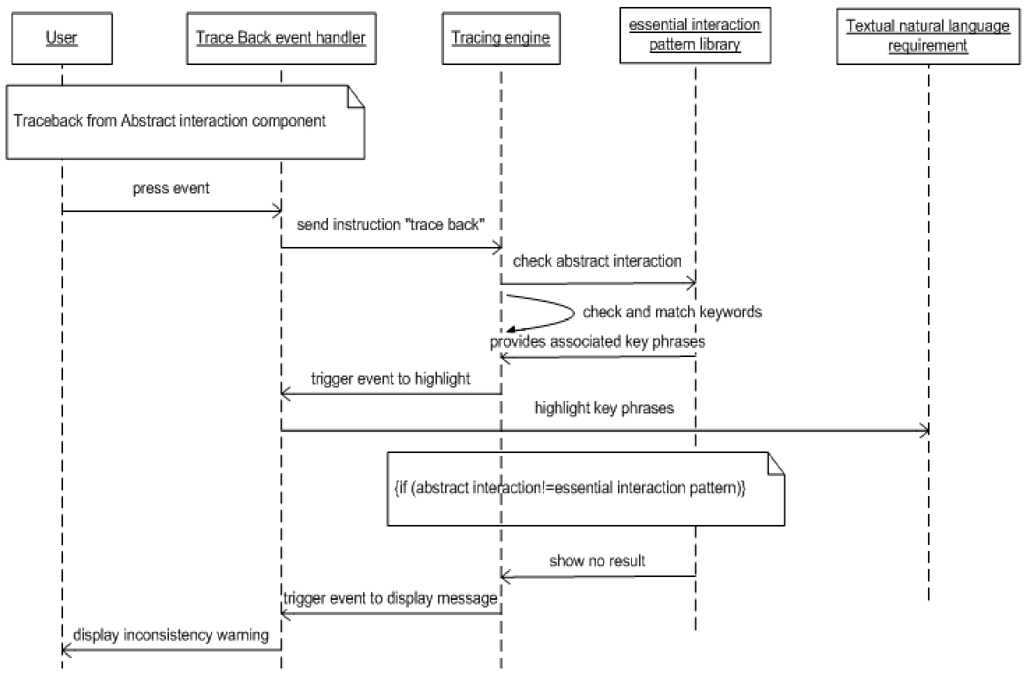

User
Trace Back event handler
Tracing engine
essential interaction pattern library
Textual natural language requirement
Traceback from Abstract interaction component
press event
send instruction "trace back"
check abstract interaction
check and match keywords
provides associated key phrases
trigger event to highlight
highlight key phrases
{if (abstract interaction!=essential interaction pattern)}
show no result
trigger event to display message
display inconsistency warning

Figure 5. 12: Example of Trace Back interaction from Abstract Interaction

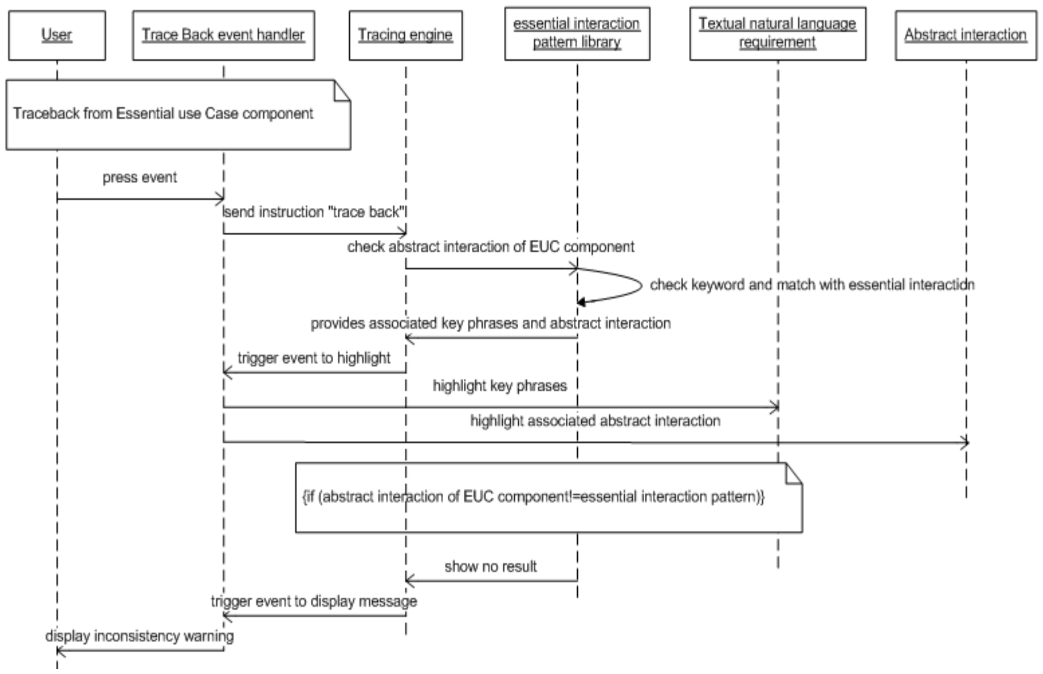

User
Trace Back event handler
Tracing engine
essential interaction pattern library
Textual natural language requirement
Abstract interaction
Traceback from Essential use Case component
press event
send instruction "trace back"
check abstract interaction of EUC component
check keyword and match with essential interaction
provides associated key phrases and abstract interaction
trigger event to highlight
highlight key phrases
highlight associated abstract interaction
{if (abstract interaction of EUC component!=essential interaction pattern)}
show no result
trigger event to display message
display inconsistency warning

Figure 5. 13: Example of Trace Back interaction from EUC component

III. Map to EUC

The event handler for mapping the abstract interaction to Marama Essential, “Map Abstract interactions to EUC”, helps to generate the Essential Use Cases automatically. The event handler works with the mapping engine to map the abstract interaction to the EUC diagram. The mapping engine analyses and matches the selected abstract interaction with the property in the interaction pattern library. Then, the abstract interaction is mapped automatically to the EUC together with its category, either user intention or system responsibility. The event handler will not map the newly added abstract interaction to the EUC component if it does not exist in the essential interaction pattern library and the textual natural language requirement is not updated. This action will also trigger an inconsistency warning to notify the inconsistency error. A sequence chart to illustrate the interaction is shown in Figure 5.14.

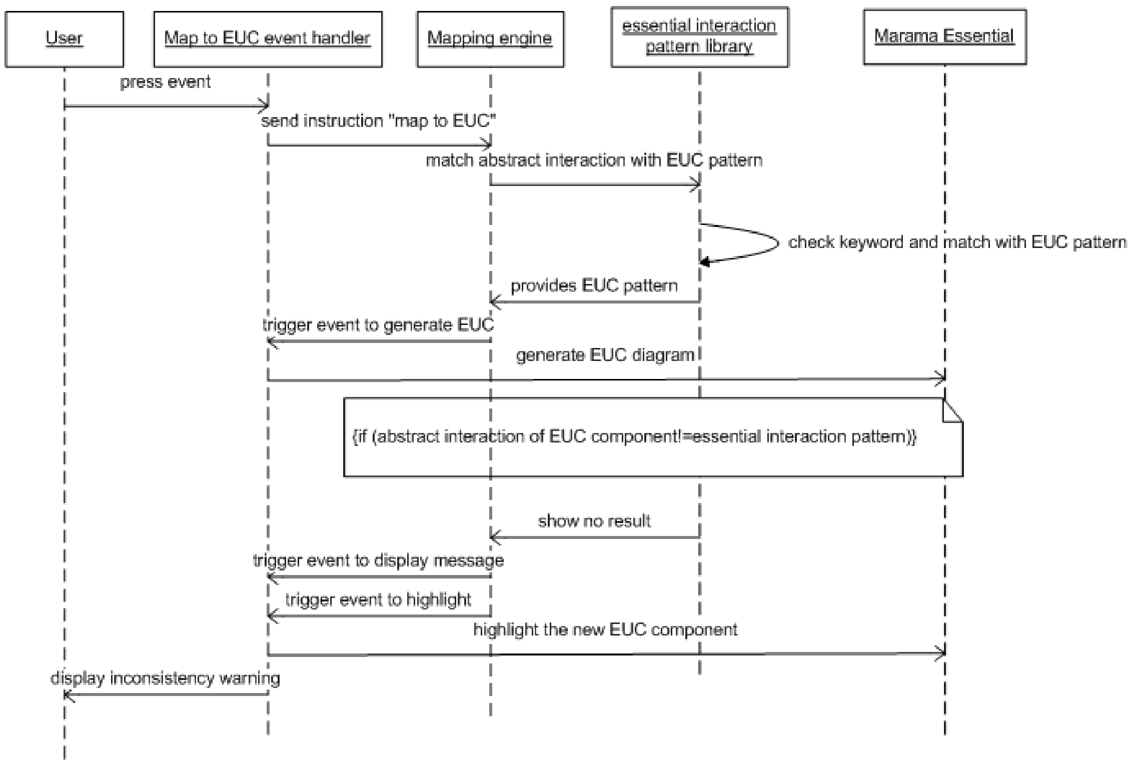
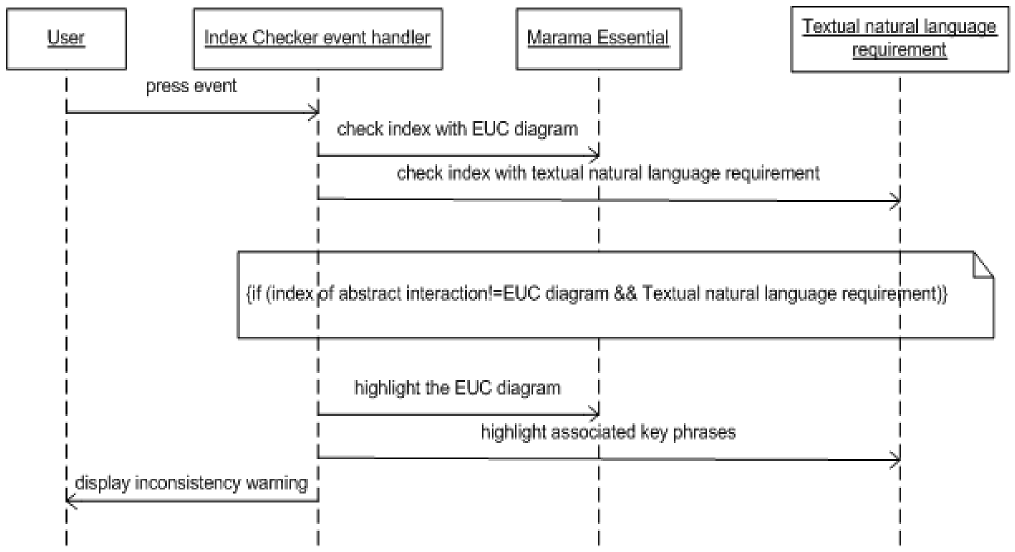

User
Map to EUC event handler
Mapping engine
essential interaction pattern library
Marama Essential
press event
send instruction "map to EUC"
match abstract interaction with EUC pattern
check keyword and match with EUC pattern
provides EUC pattern
trigger event to generate EUC
generate EUC diagram
{if (abstract interaction of EUC component!=essential interaction pattern)}
show no result
trigger event to display message
trigger event to highlight
highlight the new EUC component
display inconsistency warning

Figure 5. 14: Example of Map To EUC interaction from Abstract Interaction

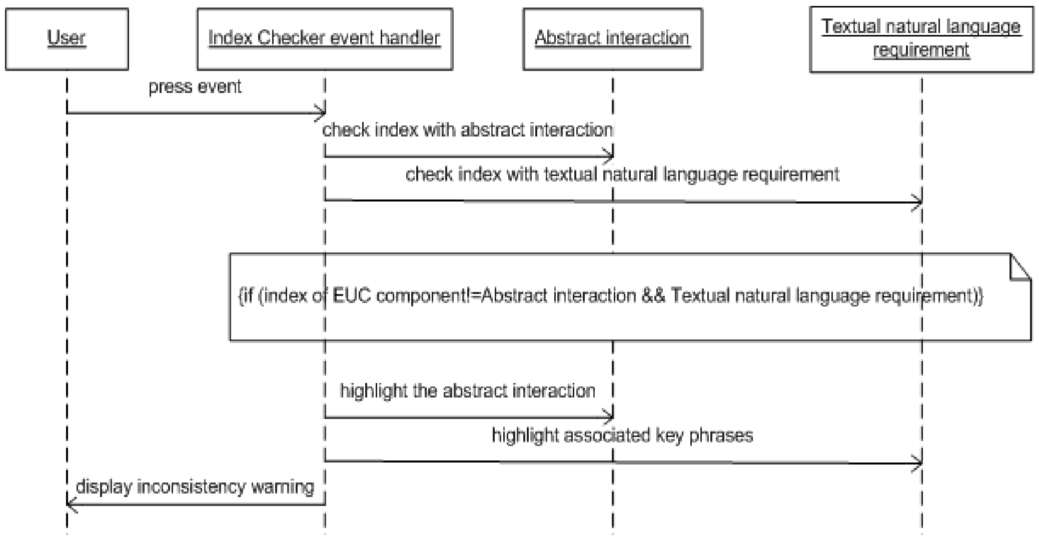
IV. Index Checker

The event handler Index Checker acts as a checker for the consistency of the sequence for both abstract interaction and EUC Diagrams in Marama Essential. The Index Checker checks the index and location for each abstract interaction and EUC component. Both need to be in sequence with ordering consistent with the textual natural language requirements. If there is any change of the sequence or location for both, the event handler provides a warning about the inconsistency that has occurred. Sequence charts to illustrate this interaction are shown in Figures 5.15 and 5.16.



User
Index Checker event handler
Marama Essential
Textual natural language requirement
press event
check index with EUC diagram
check index with textual natural language requirement
{if (index of abstract interaction!=EUC diagram && Textual natural language requirement)}
highlight the EUC diagram
highlight associated key phrases
display inconsistency warning

Figure 5. 15: Example of Index Checker interaction of Abstract Interaction



User
Index Checker event handler
Abstract interaction
Textual natural language requirement
press event
check index with abstract interaction
check index with textual natural language requirement
{if (index of EUC component!=Abstract interaction && Textual natural language requirement)}
highlight the abstract interaction
highlight associated key phrases
display inconsistency warning

Figure 5. 16: Example of Index Checker interaction of EUC component

In conclusion, all event handlers assist to automate the traceability process and help to trigger inconsistency warnings and visualise the related component if they detect any inconsistencies in any of the requirements components.

5.5 Evaluation

In order to verify the feasibility of our MaramaAI (Automated Inconsistency Checker), we have conducted a preliminary evaluation of the usefulness and the ease of use of MaramaAI by eight software engineering post-graduate students, several of whom had previously worked in the industry as developers and/or requirements engineers. All were familiar with the EUC modeling approach. Each participant was given a brief tutorial on how to use the tool and some examples of how the EUC model is derived from the textual natural language requirements. The textual natural language written in a form of user scenario of "voter registration" by Some [6] was used for this evaluation and is shown in Figure 5.17. Participants were asked to input the given scenario and then use the MaramaAI to retrieve the abstract interaction. They were also allowed to explore the event handler by tracing the abstract interaction from the textual natural language requirement, tracing back the requirement either from abstract interaction or EUC diagram and mapping the abstract interaction to the EUC Diagram in the Marama Essential. The participants were also asked to make any changes to any of the requirements, such as add, delete and change ordering, and then to observe the facilities provided by the tool in checking and detecting inconsistencies. The participants rated the usefulness and the usability of the tool together with its inconsistency detection. They also rated the consistency in textual natural language requirement, abstract interaction and EUC diagram. Our evaluation was conducted using a standard evaluation method – a Likert scale with a five part scale. It contains a set of three questions addressing each of these characteristics. The results of each corresponding three question blocks were averaged to produce the results shown in Tables 5.2 and 5.3. The type of questions for each characteristic of user perception on usefulness and the usability of the tool is in Table 5.1.

Register for vote
Primary Actor:Voter
Goal: An unregistered voter wants to register in order to be able to vote. If successful, the system generates a login id and the system generates a login id and password for the voter.
Precondition: EVote System is online
Postcondition: Voter is registered

1. Voter loads EVote system is online
2. Voter selects voter registration option
3. EVote system asks for name, social security number, date of birth, address
4. Voter provides name and social security number, date of birth, address
5. EVote system checks Voter status
6. Evote System generates Voter login id and password
7. 1.a.After 60 sec

1.a. EVote system displays time out page
2.a. After 60 sec
2.a.1. EVote system displays time out page
3.a After 60 sec
3.a.1 EVote system displays time out page

5.a. Voter data is not in record
5.a.1 Evote System displays incorrect information error page.

Figure 5. 17: The scenario “Voter Registration” [6] used for the Evaluation

User Perception Characteristics	Component	Questions
Usefulness	Abstract interaction	It is useful to capture the essential requirement (abstract interaction).
		It helps to be more effective to capture the abstract interaction.
		It makes it easier to capture abstract interaction.
	Marama Essential	It is useful to be used to display Essential Use Case.
		It helps to be more effective to capture requirements in a form of Essential Use Case.
		It makes it easier to understand the interaction in the Essential Use Case.
	Consistency Management	It is useful to detect inconsistency in the requirement.
		It helps to be more effective to manage the consistency of the requirement components: textual natural language requirement, abstract interaction and Essential Use Cases.
		It makes it easier to detect inconsistencies error in the requirement
Ease of Use	Automated Tracing Tool	It is easy to use to capture requirement by using Essential Use cases with Marama Essential
		It is user friendly.
		I don't notice any inconsistencies as I use the tool.
	Inconsistency Management	It is easy to detect the inconsistency of the requirements.
		It is user friendly
		I don't notice any inconsistencies as I use the tool.

Table 5. 1: User Perception Characteristics and Questions Evaluating Them

Table 5.2 shows the results for the evaluation result on the usefulness aspect of the tool. This shows that almost all of the participants agreed that MaramaAI is useful for finding the abstract interaction, capturing requirements using the EUC model and also for checking the inconsistency of the requirements. Overall, the usefulness of finding abstract interactions by using our tool is almost 94%, where 56.3% identified it as very useful and 37.5% identified it as always useful. A further 6% of the participants felt that it was sometimes useful to extract the abstract interaction automatically, primarily because the tool might be constrained by the domains available in the essential interaction pattern library. It was identified in the evaluation that approximately 94% of participants agreed that using MaramaAI with the Marama Essential model was useful in capturing requirements. About 56.3% identified it as very useful and another 37.5% identified it as always useful. The remainder, 6.2% of the participants, thought it was sometimes useful to use it as a tool to capture requirement as they were more familiar with using UML diagrams than Essential Use Case diagrams. For the consistency management support, approximately 94% agreed that the tool provided useful inconsistency checking and maintained the consistency of the requirements. About 56.3% of participants thought it was very useful and around 37.5% felt that the tool was always useful in managing the consistency. Again, the remainder, or 6.2% felt it was only sometimes useful in managing the consistency as they would have liked to have more complex consistency checking by the tool. However, all participants agreed that the tool assisted them to save time in capturing requirements and to manage the consistency issue between the requirements.

Category	Abstract Interaction (%)	Marama Essential (%)	Consistency Management (%)
Very Useful	56.3	56.3	56.3
Always useful	37.5	37.5	37.5
Sometimes Useful	6.2	6.2	6.2
Little useful	0	0	0
Not Useful	0	0	0

Table 5. 2: Tool Usefulness Result

Category	Automated Tracing Tool (%)	Inconsistency Management (%)
Very Easy	59.4	56.3
Always Easy	37.5	37.5
Sometimes Easy	3.1	6.2
Seldom Easy	0	0
Not Easy	0	0

Table 5. 3: Tool Ease of Use Results

The ease of use of the MaramaAI was also evaluated and the results are presented in Table 5.3. Both tracing and inconsistency checking features were evaluated. All participants agreed that both components were user friendly and easy to use in the example tasks performed. For the MaramaAI, approximately 95% agreed that the tool was easy to use, about 59% agreed that the tool was very easy to use and almost 37.5% agreed that the tool was always easy to use. Most thought that the event handlers were easy to use and really helped to automate the process. Only about 3% felt it was only sometimes easy to use. This small had difficulty in understanding the layout used by MaramaAI. The participants were confused with the shapes and colour used to represent the requirement components. For inconsistency checking of the requirements, almost 94% agreed that it was easily handled and understood. Approximately 56% agreed it was very easy to handle and another 37.5% agreed it was always easy to handle. Only 6.2% of the participants thought it was sometimes easy to check the inconsistency, because the tool currently just provides a warning on the detected inconsistency and there is no way of resolving it automatically. This minority group also wanted the tool to have an inconsistency warning together with the feedback. Table 5.4 shows further the feedback received from the participants.

Participants	Feedback
1.	"I don't really understand the layout of tool". But, overall the tool looks OK. "
2.	None.
3.	"No feedback is given with the warning. Need help to resolve the inconsistency""
4.	"OK""
5.	"Other color used for the shapes""
6	None
7.	"Didn't found this type of tool before. Would like to have more complex consistency management"
8.	"Easy to use"

Table 5. 4: Participants Open Feedback

5.6 Summary

We have discussed our approach and the advantages of using a traceability technique together with a semi formal specification in the form of an Essential Use Cases (EUC) to manage consistency among and between textual natural language requirements, abstract interactions and EUC. Traceability and consistency between these artefacts are visualised with the support of Marama. We described a proof of concept support environment, MaramaAI, that generates tracing and mapping among textual natural language requirements, abstract interactions and EUCs. Our tool is also shown to assist users and requirements engineers to capture requirements and generates EUCs automatically. In addition, our promising preliminary evaluation results conducted on the tool's usefulness and ease of use support the assertion that MaramaAI is able to minimise human intervention in checking consistency. However, there were also minor negative results and feedback gained from the study. This motivates us to improve our tool with better support for the tool usability and consistency checking. Our next focus is to provide higher level consistency between textual natural language requirements, abstract interactions and EUCs and to further check the other requirement qualities such as correctness and completeness, using our essential interaction patterns and EUC interaction patterns. This is described in the next chapter.

Chapter 6: Requirements Quality Checking

This chapter describes a higher level consistency checking technique supporting consistency of management between representations of textual natural language requirements, abstract interaction and EUCs as well as supporting further requirement quality checking for both correctness and completeness of requirements. This is an extension to improve our previous focussed work to support translation among the three forms of requirements: textual natural language, abstract interactions, and Essential Use Case models, and then the low-level management of consistency for these three forms of requirement components (described in Chapter 5).

6.1 Introduction

As described in previous chapters, inconsistency of requirements happens when there are conflicting requirements and/or the captured requirements from stakeholders are internally inconsistent when two or more components overlap and are not aligned [21], [22], from incorrect actions [14] or from requirements clashes and bad dependencies [20]. These complications also often introduce *incomplete requirements* that are missing key definitions and constraints for the software system. *Incorrect requirements* may also occur when the requirements captured do not accurately reflect the actual requirements and needs of stakeholders. These quality problems of inconsistent, incomplete and incorrect requirements lead to development delay and various quality errors, and raise the cost of the system development process, which often risks the success of the overall project [22].

To address this problem, researchers have produced various approaches, either heavyweight or lightweight, to support the requirement quality issue, but they mainly focus on consistency [17, 94, 127, 135]. As described in the previous chapter, we also manage the consistency issue by applying a lightweight approach using traceability with automated tool support. However, there are still some limitations which we need to overcome. Thus, the gaps identified and feedback gained from our previous work have motivated us to extend our work to check at a higher level consistency together with other requirement qualities, such as completeness and correctness, by using the concept of essential interaction patterns and EUC interaction patterns. These concepts assist in detecting potential quality problems, especially inconsistencies, incompleteness and incorrectness.

The essential interaction pattern library contains a list of important key phrases (essential interactions) and mappings to appropriate essential requirements (abstract interactions) which support a variety of different application domains. Essential interactions are not categorised based on one particular scenario but can be associated with multiple scenarios, such as online booking, e-commerce, online business, online banking, an online voting system, online reservation and mobile applications. Thus, multiple essential interactions from various domains can be associated with one well-defined abstract interaction. This approach allows low-level requirements problems to be identified; for example, identification of phrases of textual natural language requirements with no corresponding EUC abstract interactions or identification of EUC interactions added by the requirements engineer without any textual natural language requirement phrase(s). The essential interaction pattern library and its usage were described in detail in Chapter 4. In this chapter we will focus only on the elaboration of the EUC interaction pattern.

6.2 EUC interaction pattern

A key reason we chose to use the EUC model is that it also lends itself to a deeper analysis, enabling identification of potential problems with the extracted requirements. A set of “best practice” EUC interaction patterns or templates can be identified for a range of typical user/system interactions in a wide variety of domains [39]. Biddle et al.[39] provide a set of styles or patterns which need further enhancements to be made by the user when writing EUCs. Their aim is to allow a user to write a good EUC more quickly. They also developed a tool called UKASE, a web-based use case management tool to support the reusability of EUCs [43]. Although both works focus on supporting the generation of an EUC, neither focuses on using the EUC patterns for consistency as well as completeness and correctness checking work. This gap motivates us to check for these problems in the extracted EUCs. To do this, an EUC interaction pattern library is developed.

As described earlier, the textual natural language specifications we use are described in the form of a user scenario or story. Therefore, the EUC interaction library stores the ‘best practice” interactions of the EUC for each set of scenarios or use case stories. Table 6.1 illustrates the examples of EUC interaction patterns for scenarios or use case stories such as “reserve item”, “purchase item”, “make a transaction”, ”book item” and “make a registration” with their sequences of abstract interactions.

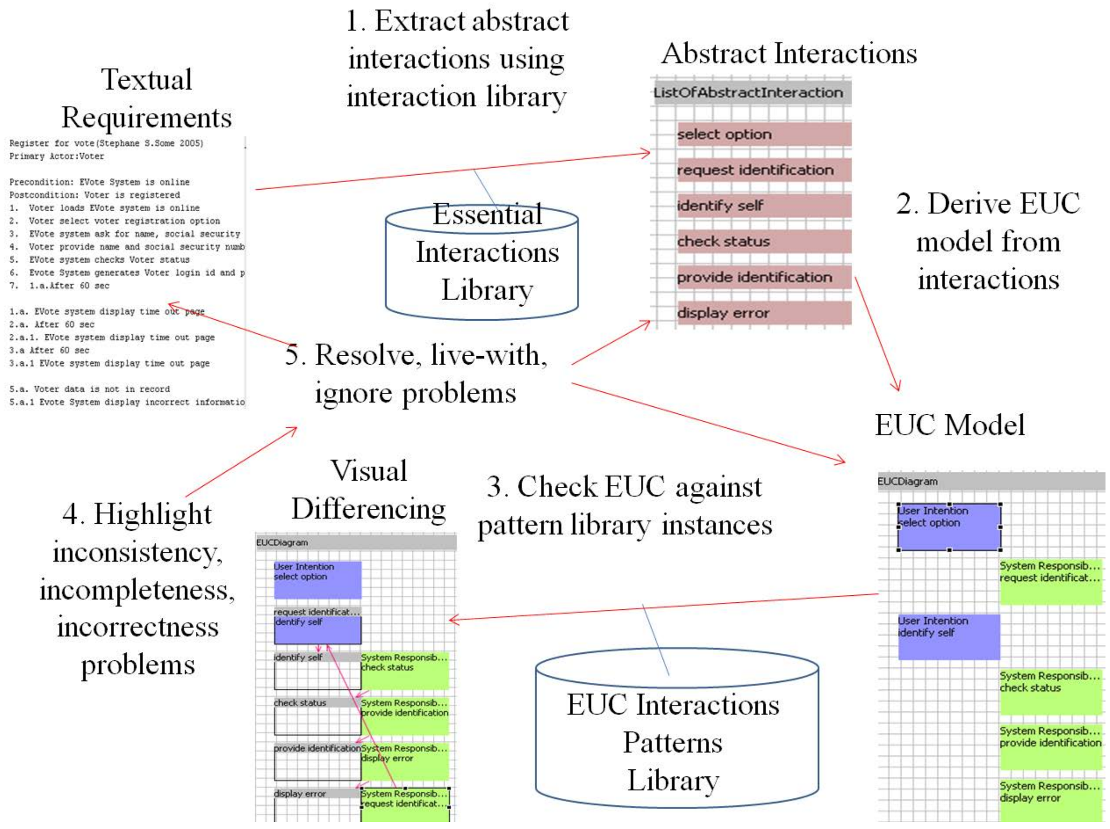
Scenarios/ Use Case stories	User intention Abstract Interaction	System responsibility Abstract Interaction
Reserve item	choose	
		offer choice
		view detail
		request identification
	identify self	
		confirm booking
Purchase item	choose	
		check status
	identify self	
	provides detail	
		verify identity
		request confirmation
		view detail
Make a transaction	select option	
	choose	
	select amount	
		verify identity
		print
Book item	identify self	
	select option	
	select item	
	insert information	
		Print
Make a registration	select option	
		request identification
	identify self	
		check status
		provide identification
		display error

Table 6. 1: Examples of EUC Interaction Patterns

Once an EUC model has been extracted, it can be compared against a pattern in our EUC Interaction Pattern Library. An extracted EUC model would be expected to conform to one of the patterns, or templates, in this library. If it deviates from this pattern, this typically indicates incompleteness (missing interactions), incorrectness (redundancy, wrongly sequenced interactions or wrong interactions), and possible inconsistency (conflicting or nonsensical interactions).

6.3 Our Approach

We have applied the EUC interaction pattern library concept together with an inter-representational traceability approach to check for requirements quality problems (inconsistency, incompleteness or incorrectness) that exist in any of the requirement representation components; textual natural language, abstract Interactions and Essential Use Cases (EUC). Figure 6.1 shows an outline of our requirements' quality management process.


Textual Requirements
1. Extract abstract interactions using interaction library
Abstract Interactions
ListOfAbstractInteraction
select option
request identification
identify self
check status
provide identification
display error
Essential Interactions Library
2. Derive EUC model from interactions
5. Resolve, live-with, ignore problems
EUC Model
4. Highlight inconsistency, incompleteness, incorrectness problems
Visual Differencing
3. Check EUC against pattern library instances
EUCDiagram
User Intention select option
request identificat...
identify self
identify self
System Responsib... check status
check status
System Responsib... provide identification
provide identification
System Responsib... display error
display error
System Responsib... request identificat...
EUC Interactions Patterns Library
EUCDiagram
User Intention select option
System Responsib... request identificat...
User Intention identify self
System Responsib... check status
System Responsib... provide identification
System Responsib... display error

```
Register for vote(Stephane S.Some 2005)
Primary Actor:Voter

Precondition: EVote System is online
Postcondition: Voter is registered
1.  Voter loads EVote system is online
2.  Voter select voter registration option
3.  EVote system ask for name, social security
4.  Voter provide name and social security numb
5.  EVote system checks Voter status
6.  Evote System generates Voter login id and p
7.  1.a.After 60 sec

1.a. EVote system display time out page
2.a. After 60 sec
2.a.1. EVote system display time out page
3.a After 60 sec
3.a.1 EVote system display time out page

5.a. Voter data is not in record
5.a.1 Evote System display incorrect informatio
```

Figure 6. 1: Outline of our Requirement Quality Management Process

Illustrated in Figure 6.1 is the outline of our process for managing requirements quality. Textual natural language requirements are first translated into a set of abstract interactions (1). This is done by using our essential Interactions library of concrete abstract interaction mappings, which abstract common expressions and phrases into EUC abstract interactions. These abstract interaction sequences are then translated into an EUC model to capture the requirements (2). This is done by applying EUC structuring rules to the interactions and a visual EUC requirements model is then generated. All of this is as per the processes described in Chapter 5. A set of inter-model checks between different requirements representation components and intra-model checks of each specific model is then conducted (3). The sequence of EUC interactions is compared to common sequences, or EUC interaction patterns, in our EUC interaction patterns library. The extracted EUC model's abstract interactions are thus compared to an expected essential interaction and EUC pattern's set of abstract interactions and their sequencing. These comparison processes highlight the potential intra- and inter-model problems such as the following.

- *Sequencing of requirement elements:* The sequence of abstract Interactions and EUC components must be in the same order as the sequence of essential interactions in the textual natural language requirement. This detects inconsistencies between models where one has been edited and others not. The ordering of the interactions between user and the system also needs to be consistent.
- *Naming of requirement elements:* The name of an EUC component must be the same as the abstract interaction and these need to map to a specific essential interaction in the textual natural language requirement. The abstract interaction also needs to match one of the abstract interactions in the essential interaction pattern library. This detects inconsistencies between models and also incompleteness. This is to ensure the completeness and correctness of the textual natural language requirements, and to maintain consistency between the abstract interaction, textual natural language requirement and the EUC diagram.
- *Consistency with changing components*: All three requirements' representations - textual natural language scenario, abstract interaction and EUC - must be consistently updated if elements in any one of the models are modified by the requirements engineer. Modification processes include adding, deleting, re-sequencing and changing properties of elements.
- *Consistency within models:* The EUC and abstract interaction sequence semi-formal notations have meta-models with constraints expressed over them, allowing low-level validation of correctness and internal notation consistency. These check for low-level intra-notation consistency, completeness and correctness. For example, the EUC has start/end interactions, naming conventions of elements are met and all elements are part of a valid sequence of EUC model-compliant interactions.
- *EUC interaction pattern matching:* the abstract interaction elements and the sequence of elements in EUC models need to match a suitable template in the EUC pattern library. Updating an abstract interaction or EUC element to conform to matching components requires updating the equivalent in the textual natural language requirement representation based on the matching pattern in the EUC pattern library. This detects incomplete and incorrect requirements elements. EUC models not conforming to a recognised pattern usually indicate missing, duplicated or redundant elements, or incorrectly expressed interaction components and sequences in the extracted requirements.

When problems with requirements models are detected, we focus on providing warning, feedback notification and visualisation of the quality issues existing in any component (4). Components that mismatch, do not exist in one model, have differing sequencing between components, or that overlap with non-corresponding names or other information, are classed as an "inconsistency". Detected redundancy of a component or a mismatch between a component and the expected element in an otherwise matching pattern is classed as “incorrectness”. Missing components or sequences in a model compared to an otherwise matching pattern are classed as “incomplete”. The set of requirements is assumed to be “complete” [136] once all the requirements model elements satisfy a match or matches in the EUC interaction pattern library. Requirements engineers can choose to do one of the following.

I. Resolve a detected quality issue by modifying the components based on the results of the consistency engine recommendation.

II. Tolerate the inconsistency until later, with our tool tracking it.

III. Strictly ignore the inconsistency (5).

We avoid forcing requirements consistency immediately as consistency rules cannot always automatically maintain the consistency of the set of requirement components. For example, if the sequence of components of the abstract interaction or EUC is problematic, we cannot automatically enforce a change in the structure of the textual natural language as this requires manual intervention. In this situation, a warning and notational element highlighting make users aware that the inconsistency is still present. Explicitly ignoring the inconsistency (suppressing warnings) is also allowed as we respect requirements engineers to make the final decision on the quality of their requirements. End-user stakeholders can view updated and/or annotated textual requirements at any time to comment on the correctness and completeness of the requirements model. While the EUC model is arguably end-user-friendly, keeping it consistent with the textual natural language representation affords the latter human-centric views continued use through the requirements engineering process.

6.4 Tool Support & Usage

6.4.1 Tool Support

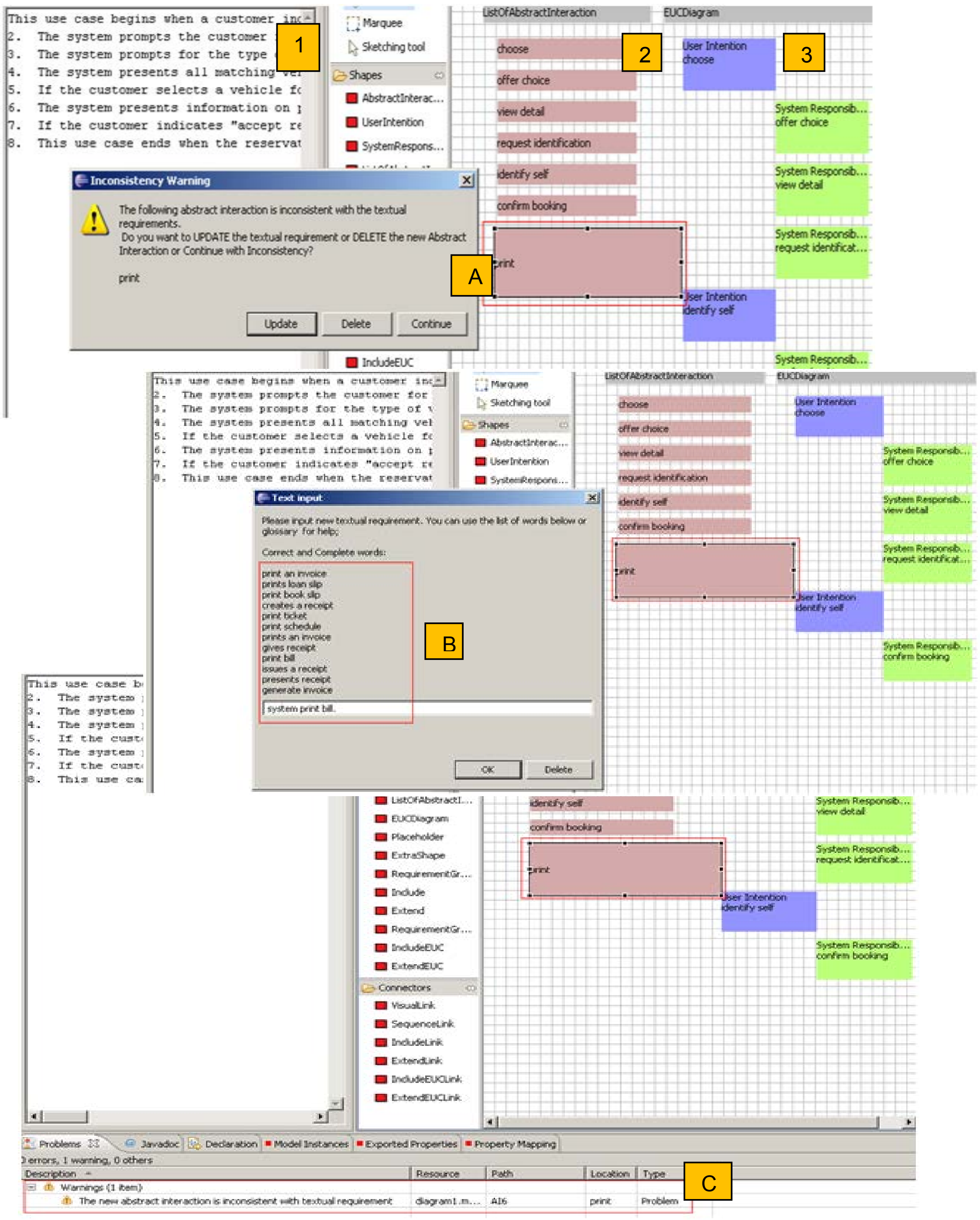
As described in Chapter 5, our prototype tool - MaramaAI (Automated Inconsistency checker) aims to help requirements engineers to manage inter-notation requirements translation and consistency management. We have extended our tool to manage a higher level consistency of requirement and to provide help for the quality improvement process based on our approach outlined in the previous section. Our tool helps to lessen the need for human intervention and minimises the time taken to manage requirements formalisation from textual natural language to the semi-formal representation in an EUC model. This is supported by the evaluation results obtained. Our tool, as discussed, not only supports incremental refinement of the requirements to address detected quality issues but also the evolution of the requirements over time. The textual natural language requirements are kept consistent with the EUC model, allowing them to co-exist during requirements engineering. Besides capturing the abstract interactions from the textual natural language requirements, a requirements engineer can also view the simplified interactions between the user and the system in the EUC automatically. This form of interaction summary allows requirements engineers to better understand the flow of the interactions, the structure of the requirements and to view key inconsistency, incompleteness or incorrectness errors identified by the tool. Warning and feedback messages are also provided to notify the requirements engineers of quality issues detected throughout the requirements refinement and correction.

Deeper analysis for completeness and correctness checks is provided by the tool. The tool compares extracted EUC models to our set of “best practice” template EUC interaction patterns. These patterns represent valid, common ways of capturing EUC models for a wide variety of domains. Matching a substantial part of an extracted EUC model to an EUC pattern indicates potential incompleteness and/or incorrectness at the points of deviation from the pattern. These potential problems are highlighted to the engineer using visual annotations on the EUC model elements. Currently, approximately 30 generic EUC interaction pattern templates are available in the tool and an extracted EUC model is expected to match one of these and, if not, differences are highlighted. New patterns can be added as required. Extracted EUC models that differ slightly, but in ways the engineer considers reasonable, can be marked as “complete”.

6.4.2 Consistency Checking

As an example of consistency checking using the outlined approach, we use the textual natural language user scenario, reserving a vehicle to illustrate requirements extraction, checking and evolution process using our extended MaramaAI tool.

Figure 6.2 shows an example of some textual natural language requirements (1), extracted abstract interaction phrases (2), and a generated EUC model representing the requirements (3), all as per the techniques described in Chapter 5. As previously noted, once these requirements have been extracted and represented in these three forms, MaramaAI provides low-level checking of the abstract interaction sequence and EUC model internal consistency, using their defined meta-model constraints. It also supports inter-model consistency management by propagating changes made to one representation across to the other two representations.



```
This use case begins when a customer inc
2.  The system prompts the customer for
3.  The system prompts for the type of v
4.  The system presents all matching veh
5.  If the customer selects a vehicle fo
6.  The system presents information on p
7.  If the customer indicates "accept re
8.  This use case ends when the reservat
```

Inconsistency Warning

The following abstract interaction is inconsistent with the textual requirements.
Do you want to UPDATE the textual requirement or DELETE the new Abstract Interaction or Continue with Inconsistency?

print

Update | Delete | Continue

Text input

Please input new textual requirement. You can use the list of words below or glossary for help;

Correct and Complete words:

print an invoice
prints loan slip
print book slip
creates a receipt
print ticket
print schedule
prints an invoice
gives receipt
print bill
issues a receipt
presents receipt
generate invoice

system print bill.

OK | Delete

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
The new abstract interaction is inconsistent with textual requirement	diagram1.m...	AI6	print	Problem

Figure 6. 2: Example of extracting an EUC model then adding a new abstract interaction

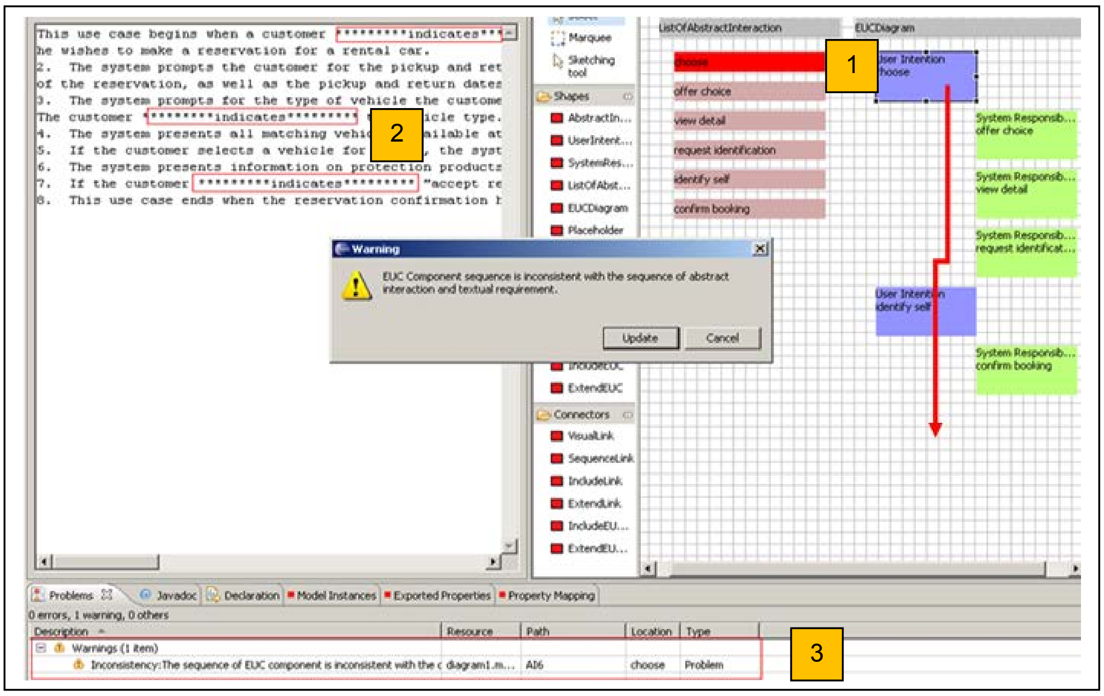
Figure 6.2 shows the addition of a new abstract interaction “print”. A warning notifies where an inconsistency is detected between representations (A). Users have the following options.

(i) Resolve the inconsistency by updating the textual natural language requirement using the provided correct and complete words (B).

(ii) Undo the change that introduced the inconsistency by deleting the new element.

(iii) Tolerate the inconsistency by ignoring it. A problem marker warning is provided to inform users about such unresolved inconsistency errors (C).

Figure 6.3 shows an example of MaramaAI tolerating inconsistency when an EUC component sequence order is changed. The EUC element “choose” has been moved to the end of the EUC model and this change affects the other two requirements forms. The textual natural language requirement and abstract interaction sequence are now inconsistent with the EUC representation. The tool colours the associated abstract interaction “choose” in red (1) and annotates the associated essential interaction “indicates” with “*****”(2). The process of detecting the inconsistency is as per described in Chapter 5. However, here, an inconsistency problem marker also appears to notify user about the inconsistency (3). Options to resolve the inconsistency by moving the associated abstract interaction element are also provided to the user. In this case, the user will have to tolerate the inconsistency until later, as changing the structure of the highlighted phrases (essential interactions) will cascade changes to the whole structure of the textual natural language requirement. Another problem marker warning is provided to continue to inform the user of the existence of an inconsistency that has not yet been resolved. The same inconsistency toleration will happen if any of the abstract interaction elements are changed to another position.



```
This use case begins when a customer *********indicates***
he wishes to make a reservation for a rental car.
2.  The system prompts the customer for the pickup and ret
of the reservation, as well as the pickup and return dates
3.  The system prompts for the type of vehicle the custome
The customer *********indicates********* t        icle type.
4.  The system presents all matching vehi        ailable at
5.  If the customer selects a vehicle for        , the syst
6.  The system presents information on protection products
7.  If the customer *********indicates********* "accept re
8.  This use case ends when the reservation confirmation h
```

Warning

EUC Component sequence is inconsistent with the sequence of abstract interaction and textual requirement.

Update Cancel

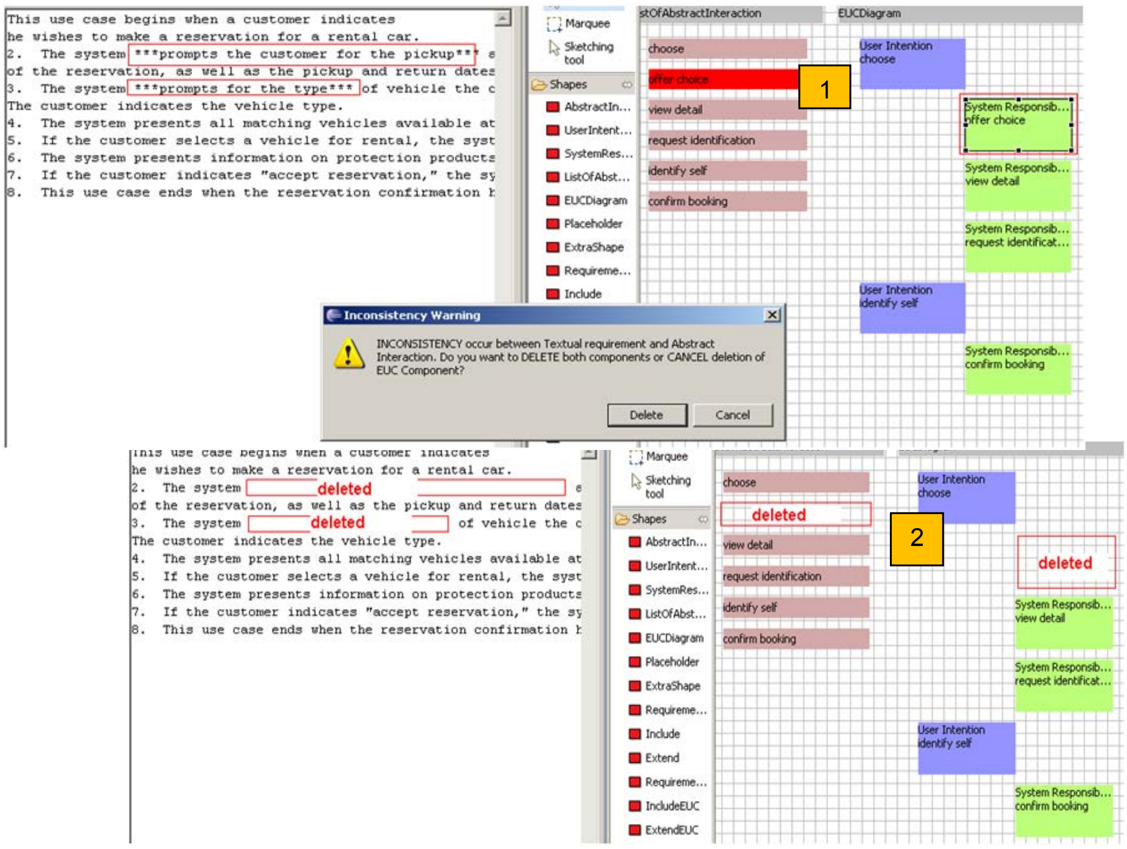
Problems | Javadoc | Declaration | Model Instances | Exported Properties | Property Mapping

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
Inconsistency:The sequence of EUC component is inconsistent with the c	diagram1.m...	AI6	choose	Problem

Figure 6. 3: Change the ordering of EUC elements

The tool also forces the user to resolve the inconsistencies if any deletion is made to either abstract interaction or the EUC component. Figure 6.4 shows that an EUC element “offer choice” is chosen to be deleted. This deletion causes an inconsistency with the abstract interaction and textual natural language requirement as indicated in Figure 6.4 (1). The abstract interaction is given a highlight and the related essential interactions in the textual natural language are also highlighted with “***”. The user only has two options: either to delete the selected element and its associated components or to cancel the deletion. If the user continues with the deletion, all the associated components will also be deleted as in figure 6.4(2). In this case, the abstract interaction element “offer choice” and the associated essential interactions “prompts the customer for the pickup” and “prompts for the type” in the textual natural language requirement are deleted.



```
This use case begins when a customer indicates
he wishes to make a reservation for a rental car.
2.  The system ***prompts the customer for the pickup*** 
of the reservation, as well as the pickup and return dates
3.  The system ***prompts for the type*** of vehicle the c
The customer indicates the vehicle type.
4.  The system presents all matching vehicles available at
5.  If the customer selects a vehicle for rental, the syst
6.  The system presents information on protection products
7.  If the customer indicates "accept reservation," the sy
8.  This use case ends when the reservation confirmation b
```

Inconsistency Warning

INCONSISTENCY occur between Textual requirement and Abstract Interaction. Do you want to DELETE both components or CANCEL deletion of EUC Component?

```
This use case begins when a customer indicates
he wishes to make a reservation for a rental car.
2.  The system deleted
of the reservation, as well as the pickup and return dates
3.  The system deleted          of vehicle the c
The customer indicates the vehicle type.
4.  The system presents all matching vehicles available at
5.  If the customer selects a vehicle for rental, the syst
6.  The system presents information on protection products
7.  If the customer indicates "accept reservation," the sy
8.  This use case ends when the reservation confirmation b
```

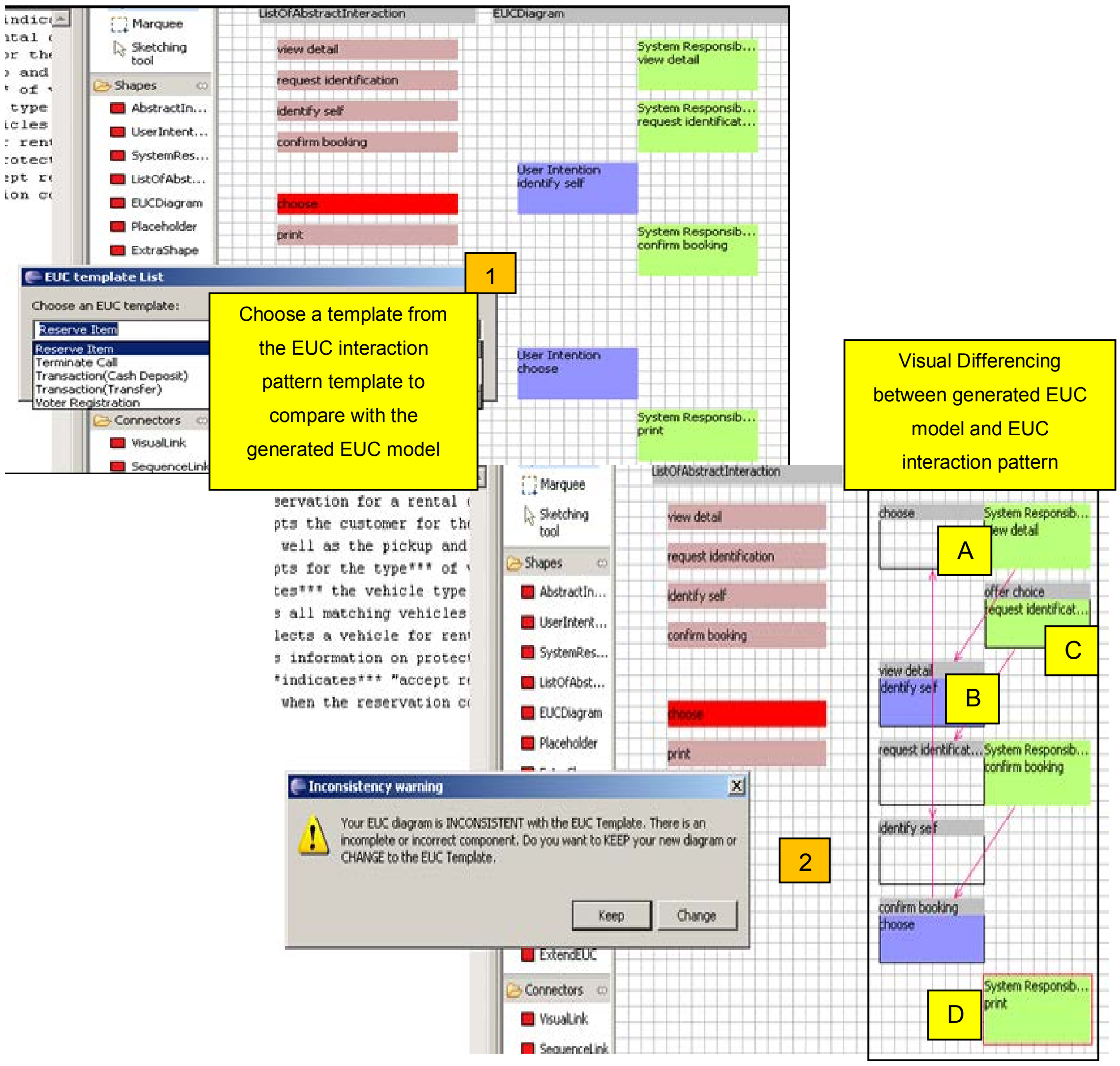
Figure 6. 4: Deletion of an EUC element

6.4.3 Inconsistency, Incorrectness and Incompleteness Checking

Further detailed analysis of the consistency, correctness and completeness of requirements models is provided by using EUC pattern library instances to validate the extracted EUC model. To do this, the checker attempts to match the extracted EUC model with one of the generic EUC interaction patterns or templates in the EUC interaction pattern library. Currently, there are approximately 30 generic EUC interaction pattern templates covering various domains developed by us and collected from the research of Constantine and Lockwood [4] and Biddle et al. [39]. The generic template is assumed to be the correct and complete interaction (an oracle) for a specific scenario. This provides the requirements engineer with a further, higher level, check of his requirement's model by comparing his EUC, representing a semi-formal model of the original textual natural language requirements, with a template which matches a "best practice" EUC representation for the abstract interaction scenario. As discussed above, this technique allows us to detect:

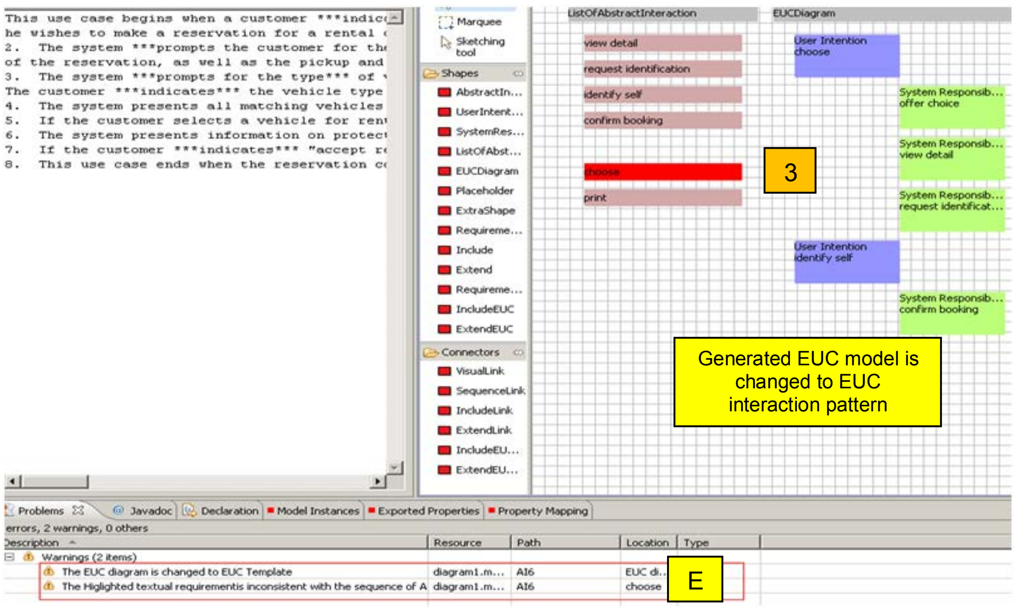
- intra-model inconsistencies (e.g. one or more unexpected abstract interactions or interactions out of expected sequence appearing in the extracted EUC model);
- incompleteness (missing interactions occur in the extracted EUC model compared to the generic template matched in the EUC pattern library); and to some degree,
- incorrectness: requirements captured in the extracted EUC model do not match a best-practice template in the pattern library, indicating possible incorrect textual requirements.

For example, Figure 6.5 shows the requirement describing reservation of a rental vehicle from a company. To check for consistency of this requirement, the user can choose a provided EUC interaction pattern template ”reserve item” (outlined in Figure 6.5(1)) to compare to the extracted EUC model. Alternatively, he can have MaramaAI compare the extracted EUC model to all available patterns and find a “best fit”, highlighting any differences from the best fit template as possible problems. MaramaAI checks whether or not the extracted EUC requirements model is consistent with the identified EUC interaction pattern library template. If differences are found, a warning message is provided and the tool uses a visual differencing approach [137] to highlight potential inconsistency, incompleteness and/or incorrectness errors that may exist in the requirements model, as shown in Figure 6.5 (2).


1
Choose a template from the EUC interaction pattern template to compare with the generated EUC model
EUC template List
Choose an EUC template:
Reserve Item
Terminate Call
Transaction(Cash Deposit)
Transaction(Transfer)
Voter Registration
2
Visual Differencing between generated EUC model and EUC interaction pattern
A
B
C
D
Inconsistency warning
Keep
Change

Your EUC diagram is INCONSISTENT with the EUC Template. There is an incomplete or incorrect component. Do you want to KEEP your new diagram or CHANGE to the EUC Template.

Figure 6. 5: Example of EUC interaction pattern template (1) and Visual differencing (2)



```
This use case begins when a customer ***indic
he wishes to make a reservation for a rental
2.   The system ***prompts the customer for th
of the reservation, as well as the pickup and
3.   The system ***prompts for the type*** of
The customer ***indicates*** the vehicle type
4.   The system presents all matching vehicles
5.   If the customer selects a vehicle for ren
6.   The system presents information on protec
7.   If the customer ***indicates*** "accept r
8.   This use case ends when the reservation c
```

```
errors, 2 warnings, 0 others
Description                                                    Resource      Path   Location  Type
⊟ Warnings (2 items)
   The EUC diagram is changed to EUC Template                  diagram1.m... AI6    EUC di...
   The Higlighted textual requirementis inconsistent with the sequence of A  diagram1.m... AI6  choose
```

Figure 6. 6: Change generated EUC model following the EUC interaction pattern template

Here, EUC interaction pattern elements are shown as a set of grey elements adjacent to the extracted EUC model. Visual link "→" annotations connect corresponding elements in the extracted EUC and EUC interaction pattern. The tool is able to show errors such as wrong sequence ordering, redundancy, missing elements and the existence of extra elements in the EUC model. Incorrect sequences are obvious via crossed links (e.g. the out of order "view detail" abstract interaction). Incorrect interaction is also shown by a grey element (e.g. "choose") near the extracted EUC model (e.g. "view detail") (A). Both are from different interaction category where element "choose" is from User Intention and "view detail" is from System Responsibility. The position is also supposed to be held by" choose" and not "view detail". Unmatched items in the EUC interaction pattern template (e.g. "view detail") or in the extracted EUC (e.g. "identify self") are highlighted (B) (in this case juxtaposed to indicate the EUC interaction pattern element could sensibly replace the extracted element). The error is the incorrect position hold by "identify self" where the position should be hold by "view detail". The incomplete item is shown by the grey shape "offer choice" overlapping the green shape "request identification" (C). This also shows that the incorrect position held by the extracted EUC component "request identification": it should actually be after the element "offer choice" The extra element "print" is highlighted with a red box (D) to show the unnecessary existence of an element in the diagram.

Based on the visualised errors, requirements engineers can choose to: change their EUC model to conform to the template view, incorporate some of the recommended changes into their model, or keep their existing EUC requirements model. For example, Figure 6.6 (3) shows that if the requirement engineers choose to change his EUC model to follow the template, the EUC model will change automatically to an EUC based on the EUC pattern template. The problem marker provides a warning if the generated EUC model is kept or changes to the pattern template or if there are still inconsistencies in any of the requirement components (E). Our philosophy is to lessen the human effort and intervention in checking for potential errors but to leave the final decision to accept or reject recommendations to the user. Our belief is that combining tool automation support to identify potential requirements errors with human acceptance and cross validation better helps unearth and fix inconsistency, incompleteness and incorrectness errors [138].

6.5 Architecture and Implementation

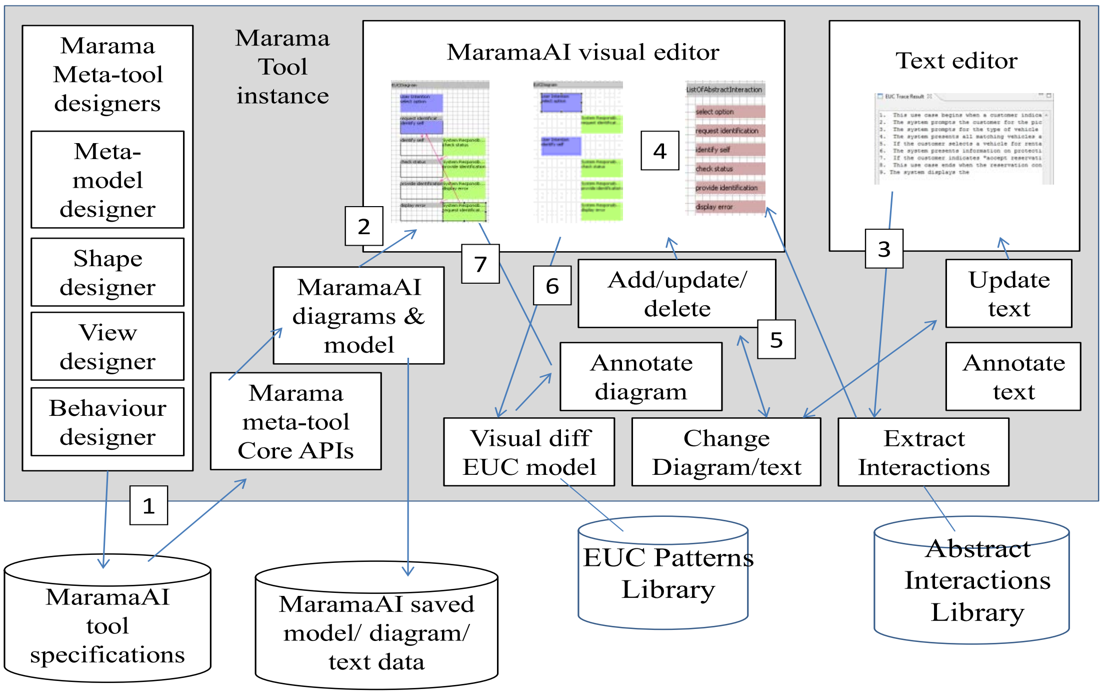


Figure 6. 7: MaramaAI tool architecture for managing consistency of requirement

We developed the extended MaramaAI toolset using our Marama meta-tools [120] and a number of specialised components for requirements extraction, analysis, comparison to the pattern library and visual differencing. An outline of the tool's architecture is represented in Figure 6.7.

I. Items (1-4) are as per described in Chapter 5. We developed the meta-model, editing tools and basic EUC model constraint management with Marama, generating a specification for the tool (1). When using MaramaAI, a requirements engineer opens the MaramaAI tool specification and the Marama meta-tool instantiates the tool including model and diagrams (2).Textual natural language requirements are extracted from plain text documents (which themselves can be extracted from Word and PDF formats). This is done by using essential interaction phrases to abstract interaction mappings in our essential interaction library (3). A list of extracted abstract interactions is generated which is then translated into an EUC model. These models are used to generate an abstract interactions list and an associated EUC diagram (4).

II. Here, we have mapped our EUC interaction pattern library approach, illustrated in Figure 1, to the consistency management framework proposed by Nuseibeh [135]. The requirements engineer can make modifications to any of the representations supported by MaramaAI (5), including changing textual representation or adding, updating, re-sequencing or removing elements in EUC or abstract interaction representations. Inconsistencies between these representations are detected and shown to the user via highlighting text and/or diagram elements. The EUC model is compared against "best practice" templates in the EUC patterns library to check its completeness and correctness (6). Differences to a chosen pattern template in the library are highlighted between the EUC model and template through visual differencing (7). This annotates the EUC model to indicate these differences. For all inconsistencies and differences from an EUC model from a pattern library template, the requirements engineer can choose to resolve the inconsistency by modifying components, to tolerate it (deferring for later attention) or to indicate his wishes to ignore the inconsistency.

Scenario	Generated EUC model with changes		EUC interaction pattern	
Reserve item		1. view detail	1. choose	
		2. request identification		2. offer choice
	3. identify self			3. view detail
		4. confirm booking		4.request identification
	5. choose		5.identify self	
		6. print		6. confirm booking

Table 6. 2: Overview of Comparing the Generated EUC model with EUC Interaction Pattern Template

Table 6.2 shows an overview of the comparison of a generated EUC model with an EUC interaction pattern based on the selected scenario template. An item with a red dashed circle shows an incorrect item that exists in the EUC model compared to the EUC interaction pattern. The blue dotted lines show an incorrect match between both models and a yellow dotted line shows an incorrect position or sequence hold by each item in the generated model compared to the EUC interaction pattern. The red dotted box with the item "offer choice" shows there is an incomplete item which is missing in the generated item compared to the EUC interaction pattern. This concept is performed using visual differencing in MaramaAI as illustrated in Figure 6.5.

III. An inconsistency is resolved by updating a representation model appropriately and MaramaAI provides support to the user by presenting and applying potential changes to resolve the inconsistency. In each case, any modification results in the models again being checked with the meta-model consistency rules and the EUC pattern template.

As mentioned earlier, we implemented the visual diagramming interfaces of MaramaAI using the Marama meta-tool [120]. This support the latter used in visual differencing. The meta-model and DSVL editors were also supplemented with event handlers to provide low-level model constraints, consistency management support and interfaces to other elements of the architecture. These were implemented in Java and include generation of dialogues and problem markers to help the user to track, tolerate and resolve inconsistencies. In Chapter 5, an event handler was described to implement extraction of textual natural language requirements into abstract interactions, and another to generate an EUC model from the abstract interaction as well as simple inconsistency checking. Three further event handlers which are written in Java and used to check for the higher level consistency checking using the essential interaction pattern and EUC Interaction Pattern, are described as below;

I. Consistency Management: This event handler is used to check any deleted item in any of the requirement components: textual natural language, abstract interaction and the EUC model. The Consistency Management checks the deleted item either in abstract interaction or an EUC component. Then the event handler will highlight the associated component in red and the corresponding textual natural language requirement with ****. The highlight components need to be deleted also, or the user is forced to cancel the deletion It also provides warning about the inconsistency that has occurred and triggers the problem marker to show the inconsistency errors. Sequence diagrams to illustrate this interaction are shown in Figures 6.8 and 6.9.

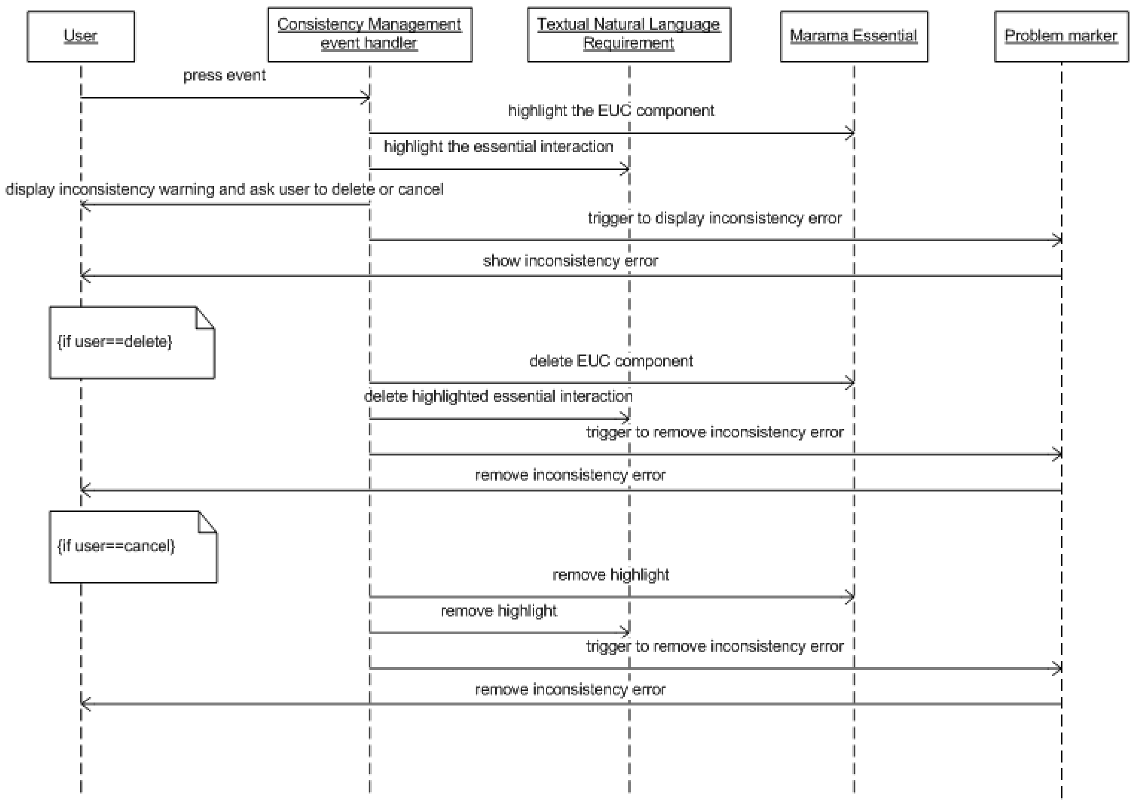

User
Consistency Management event handler
Textual Natural Language Requirement
Marama Essential
Problem marker
press event
highlight the EUC component
highlight the essential interaction
display inconsistency warning and ask user to delete or cancel
trigger to display inconsistency error
show inconsistency error
{if user==delete}
delete EUC component
delete highlighted essential interaction
trigger to remove inconsistency error
remove inconsistency error
{if user==cancel}
remove highlight
remove highlight
trigger to remove inconsistency error
remove inconsistency error

Figure 6. 8: Example of Consistency Management: Delete Abstract Interaction

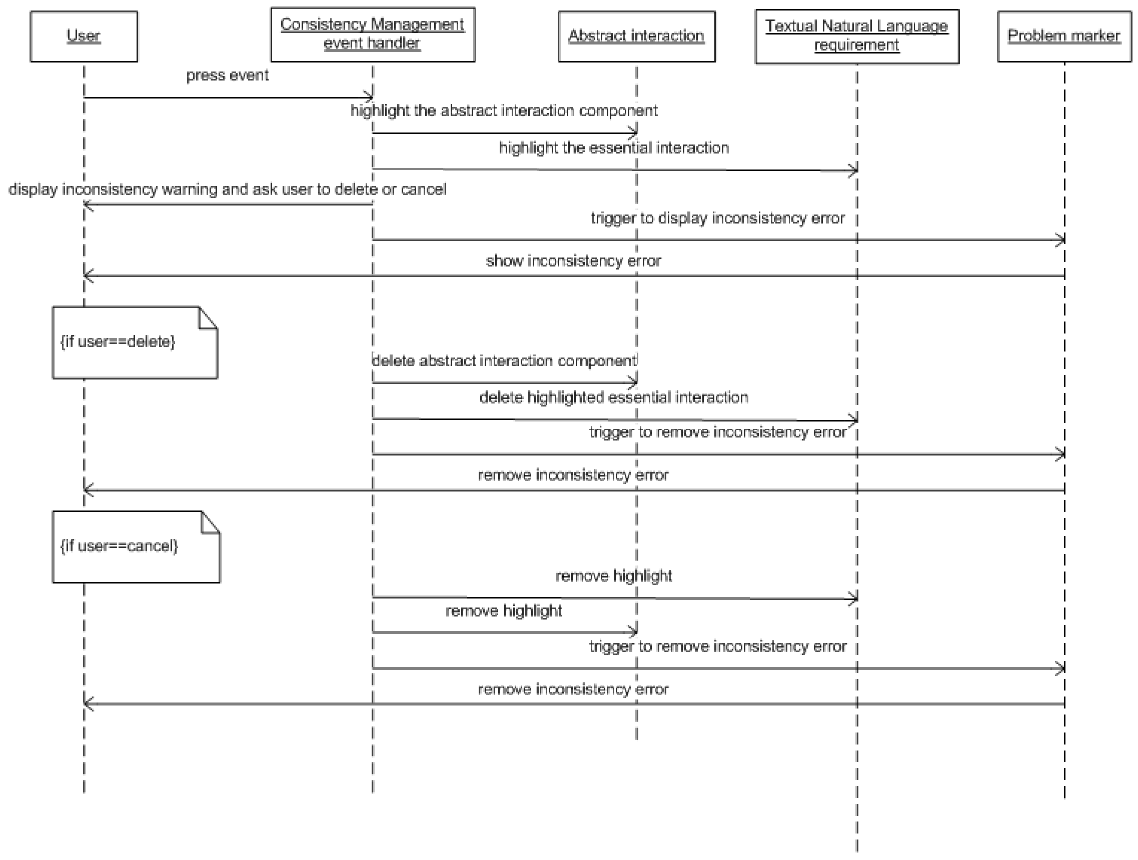

User
Consistency Management event handler
Abstract interaction
Textual Natural Language requirement
Problem marker
press event
highlight the abstract interaction component
highlight the essential interaction
display inconsistency warning and ask user to delete or cancel
trigger to display inconsistency error
show inconsistency error
{if user==delete}
delete abstract interaction component
delete highlighted essential interaction
trigger to remove inconsistency error
remove inconsistency error
{if user==cancel}
remove highlight
remove highlight
trigger to remove inconsistency error
remove inconsistency error

Figure 6. 9: Example of Consistency Management: Delete EUC Component

II. Check Consistency with a Template: This event handler is used to check and compare the generated EUC model with the EUC interaction pattern in the EUC interaction pattern library. Here, visual differencing proposed by Mehra et al. [137] is conducted to show the inconsistency, incompleteness and incorrectness in the EUC model. If any of these errors occur, the event handler provides a warning about the errors and asks the user to either change the generated EUC model to the suggested template or to live with the inconsistent model. The event handler also triggers the problem marker to show the error which still exists in the model. A sequence chart to illustrate this interaction is shown in Figure 6.10.

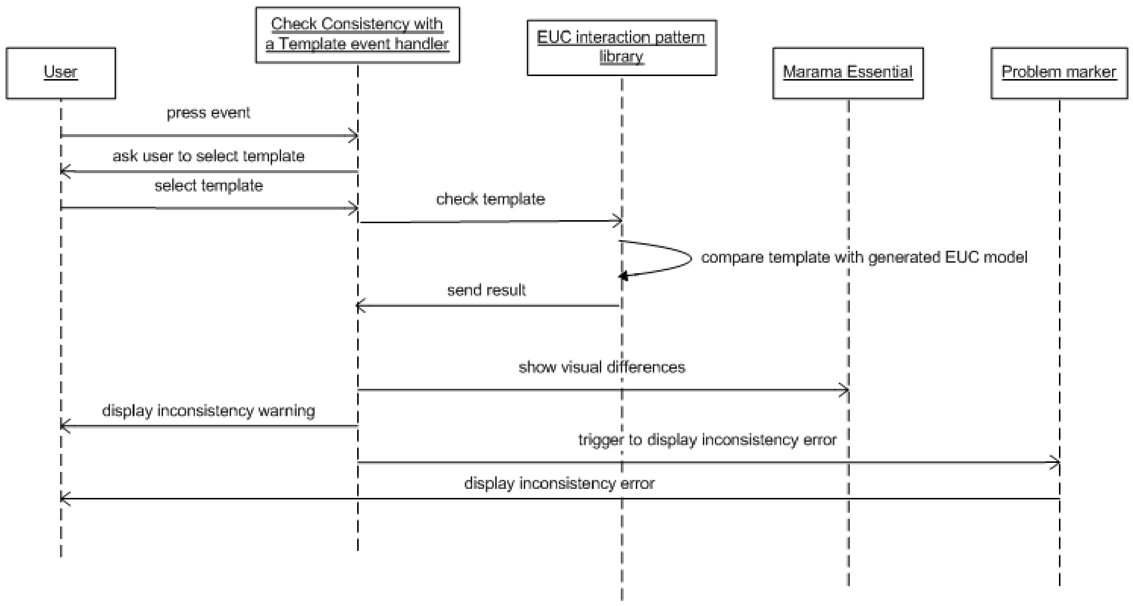

User
Check Consistency with a Template event handler
EUC interaction pattern library
Marama Essential
Problem marker
press event
ask user to select template
select template
check template
compare template with generated EUC model
send result
show visual differences
display inconsistency warning
trigger to display inconsistency error
display inconsistency error

Figure 6. 10: Example of Check Consistency with a Template for the Generated EUC Model

III. Check Keyword: This event handler is used to check the existence of new abstract interaction in either the abstract interaction component or the EUC component. The event handler checks the new abstract interaction with the textual natural language requirement and will trigger an inconsistency warning if the new abstract interaction is not matched with the essential interactions in the textual natural language requirement. It will also trigger the user to make a selection: to update the new abstract interaction with the provided suggested list of complete and correct words, to delete the new item or to continue with the addition of the new abstract interaction without any updating to the textual natural language requirement. Sequence charts to illustrate this interaction are shown in Figures 6.11 and 6.12.

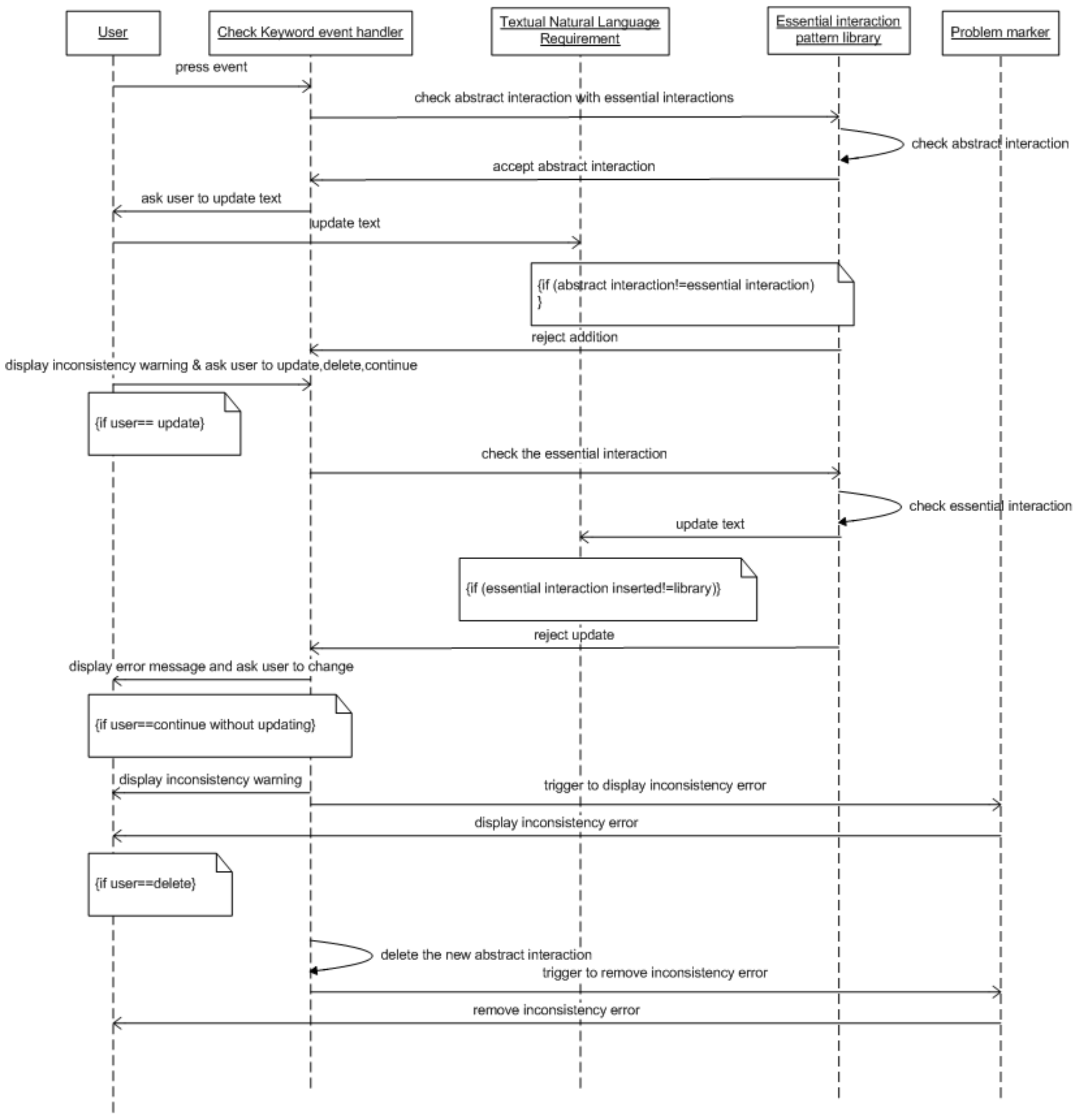

User
Check Keyword event handler
Textual Natural Language Requirement
Essential interaction pattern library
Problem marker
press event
check abstract interaction with essential interactions
check abstract interaction
accept abstract interaction
ask user to update text
update text
{if (abstract interaction!=essential interaction)
}
reject addition
display inconsistency warning & ask user to update,delete,continue
{if user== update}
check the essential interaction
check essential interaction
update text
{if (essential interaction inserted!=library)}
reject update
display error message and ask user to change
{if user==continue without updating}
display inconsistency warning
trigger to display inconsistency error
display inconsistency error
{if user==delete}
delete the new abstract interaction
trigger to remove inconsistency error
remove inconsistency error

Figure 6. 11: Example of Check Keyword of abstract interaction in the abstract interaction Component

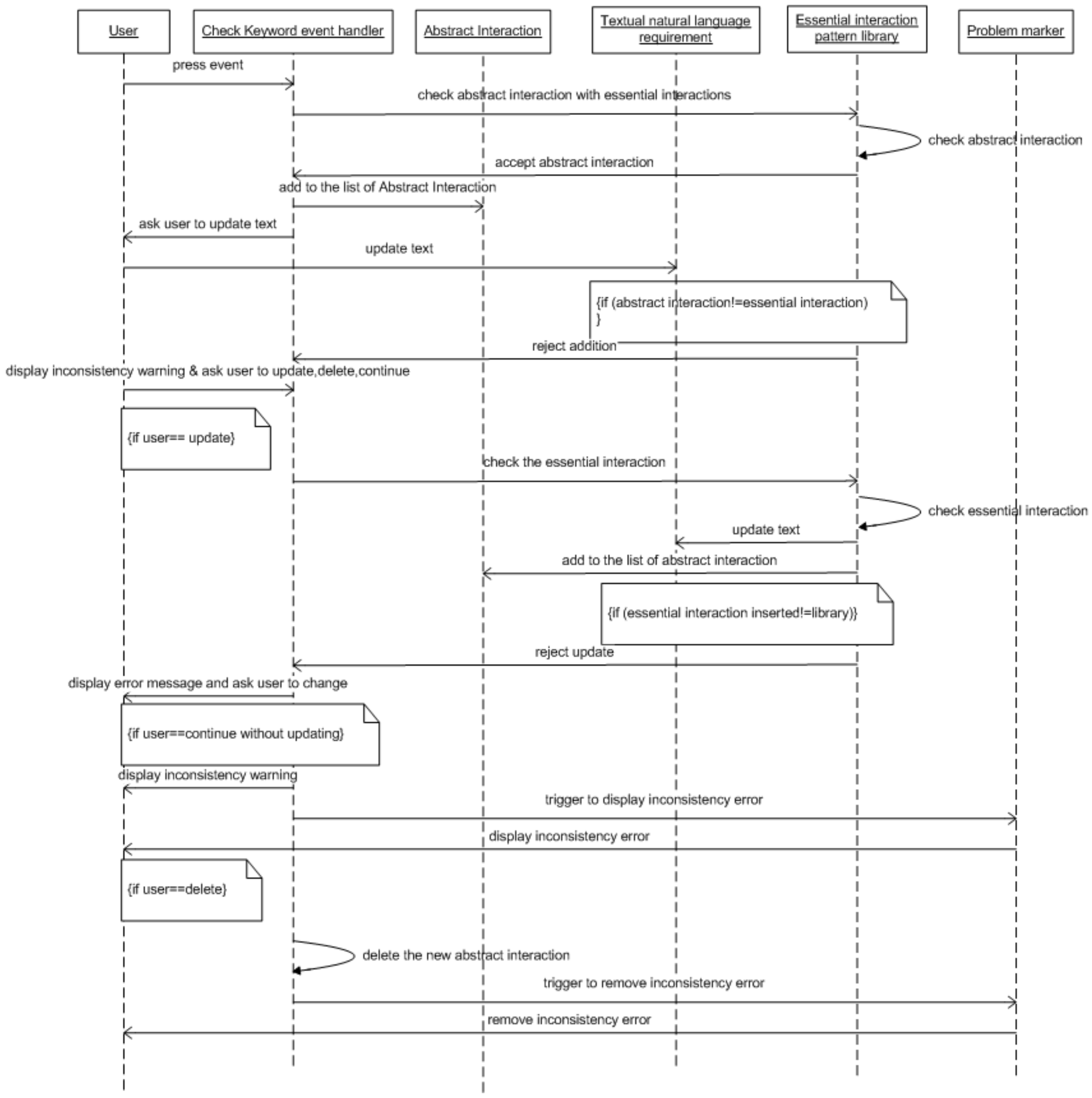

User
Check Keyword event handler
Abstract Interaction
Textual natural language requirement
Essential interaction pattern library
Problem marker
press event
check abstract interaction with essential interactions
check abstract interaction
accept abstract interaction
add to the list of Abstract Interaction
ask user to update text
update text
{if (abstract interaction!=essential interaction) }
reject addition
display inconsistency warning & ask user to update,delete,continue
{if user== update}
check the essential interaction
check essential interaction
update text
add to the list of abstract interaction
{if (essential interaction inserted!=library)}
reject update
display error message and ask user to change
{if user==continue without updating}
display inconsistency warning
trigger to display inconsistency error
display inconsistency error
{if user==delete}
delete the new abstract interaction
trigger to remove inconsistency error
remove inconsistency error

Figure 6. 12: Example of Check Keyword of abstract interaction in the EUC Component

6.6 Conclusion

In this chapter, we have described our further work in managing the requirement consistency. We described an approach supporting requirements quality improvement via a combination of semi-formal model extraction from natural language specifications and analysis using the essential interaction pattern library and an EUC interaction pattern library. Low-level inconsistency problems can be identified such as textual natural language phrases without matching semi-formal model elements and meta-model constraint violations of the extracted model. Higher-level problems, including inconsistency, incompleteness and incorrectness can be identified by comparing the semi-formal model with the essential interaction pattern and with the "best practice" examples of EUC interaction pattern templates. A visual differencing technique highlights differences between the pattern template and the EUC model. Modifications to EUC, abstract interaction and textual natural language requirements representations are also supported with consistency management support among the different representations. We have conducted the evaluation of our consistency management approach and the results are further discussed in the Evaluation Chapter, Chapter 9.
The results and feedback received from the evaluation motivate us to extend our work with end-to-end support by developing an Essential User Interface (EUI) prototype and a concrete UI view in a form-based UI from the EUC model. The generated UI could be used to verify and confirm that requirements expressed by clients are consistent with the requirement engineer's view using this alternative visualisation mechanism. This is described in the next chapter.

Chapter 7: Supporting Requirement Validation via End-to-End Rapid Prototyping

This chapter describes a significant extension of previous work which provides end-to-end rapid prototyping support for validating requirements. A new round-trip requirements engineering approach, capturing requirements as essential use cases and further translating them into “Essential User Interface” low-fidelity rapid prototypes is developed. These prototypes aid clients to better conceptualise and understand how requirements might surface in a system, enable them to provide more detailed feedback to requirements engineers and provide a complementary user-centric representation of requirements orthogonal to existing natural language and formal models. A study illustrating the challenges of requirements engineers in capturing such rapid prototypes, a tool to support requirements capture, rapid prototype generation and consistency management are also described.

7.1 Introduction

Requirements capture from clients is often difficult, time consuming and error prone [11]. Late validation, in particular, causes requirements quality to suffer [25]. This has placed a focus on techniques for early client feedback, such as use of formal and semi-formal models and heuristic algorithms [30],[139],[26] plus techniques for translating natural language requirements into such models [14, 28, 139]. While beneficial, these approaches are often difficult to use and require much effort [140], [141]. However, most clients do not understand models, formal terms or mathematics equations leading to validation problems [141],[142]. Rapid prototyping can be one of the best ways for early validation of requirements from both a requirements engineer (RE) and a client’s view respectively [143]. Using prototypes, clients gain a much clearer understanding of a proposed system via an intuitive representation, or mock-up, of the target system. This helps to reach a very early identification of missing or incorrect requirements [144],[71].

For early-stage requirements analysis, low-fidelity or abstract prototypes are useful [145]. However, developing such prototypes requires effort [71] and, Sukaviriya et al [145] assert, is poorly supported by toolsets. In chapters 5 and 6, we have developed a technique and toolset for checking consistency of requirements based on Essential Use Case (EUC) diagrams. Here, we describe a significant extension of this work providing end- to-end rapid prototyping support. EUC models are mapped to an abstract Essential User Interface (EUI) prototype model. From there they are mapped to concrete User Interface (UI) views in the form of form-based UIs. This allows the RE and client to walk-through the formalised requirements together and to validate and confirm the consistency of these requirements. We have established a set of EUI patterns for EUI prototypes and an EUI Pattern template for translating an EUI prototype to concrete UI view-HTML form and implemented them as an extension to our previous work.

7.2 Background

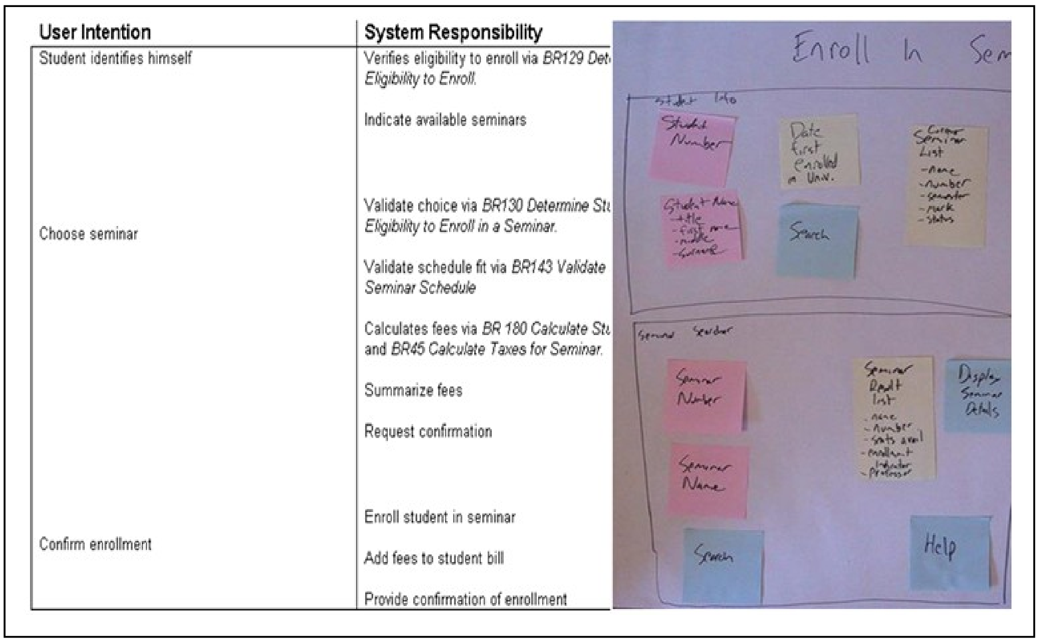
7.2.1 Rapid Prototyping

Rapid prototyping assists the requirement elicitation process by gaining early feedback from clients on captured requirements [71],[146]. Low-fidelity or abstract prototypes (often paper) are commonly used in this process [147]. Types of abstract prototypes include EUI prototypes [4], abstract user interfaces [148] or UI prototypes [149]. These are easy-to-change mock ups which encourage iteration of the elicitation and validation process [71],[149]. They allow a rough walk-through of user tasks before needing to factor in hardware or technology concerns [146] and can avoid clients being fixated at an early stage on concrete product appearance rather than functionality [71].

7.2.2 Essential User Interface (EUI) prototyping

EUI prototyping is a low-fidelity prototyping approach [10]. It provides the general idea behind the UI but not its exact details. It focuses on the requirements and not the design, representing UI requirements without the need for prototyping tools or widgets to draw the UI [150]. EUI prototyping extends from, and works in tandem with, the semi-formal representation of EUCs, both focusing on users and their usage of the system, rather than on system features [11]. It thus helps to avoid clients and REs being misled or confused by chaotic, rapidly evolving and distracting details.

Figure 7.1, from Ambler [10, 11], shows an example of an EUI prototype being developed from an Essential Use Case (EUC). The post-it notes represent abstractions of user interfaces. The different colours of these notes represent different UI elements. Pink notes represent the input field, yellow notes represent display only and blue notes represent actions [11]. Here, the RE is capturing the user intention/system responsibility dialogue represented in the EUC as possible UI functionality at a high level of abstraction. Although EUI prototyping has advantages, it has not been rigorously applied in practice as no tool support is available for such an approach. Being a whiteboard/paper technique, it does not integrate well with other tools used in the software engineering process [10]. Also, previous work has shown that the application of low-fidelity techniques in practice proves challenging [71]. Overcoming these problems was the motivation for the work described in this chapter.



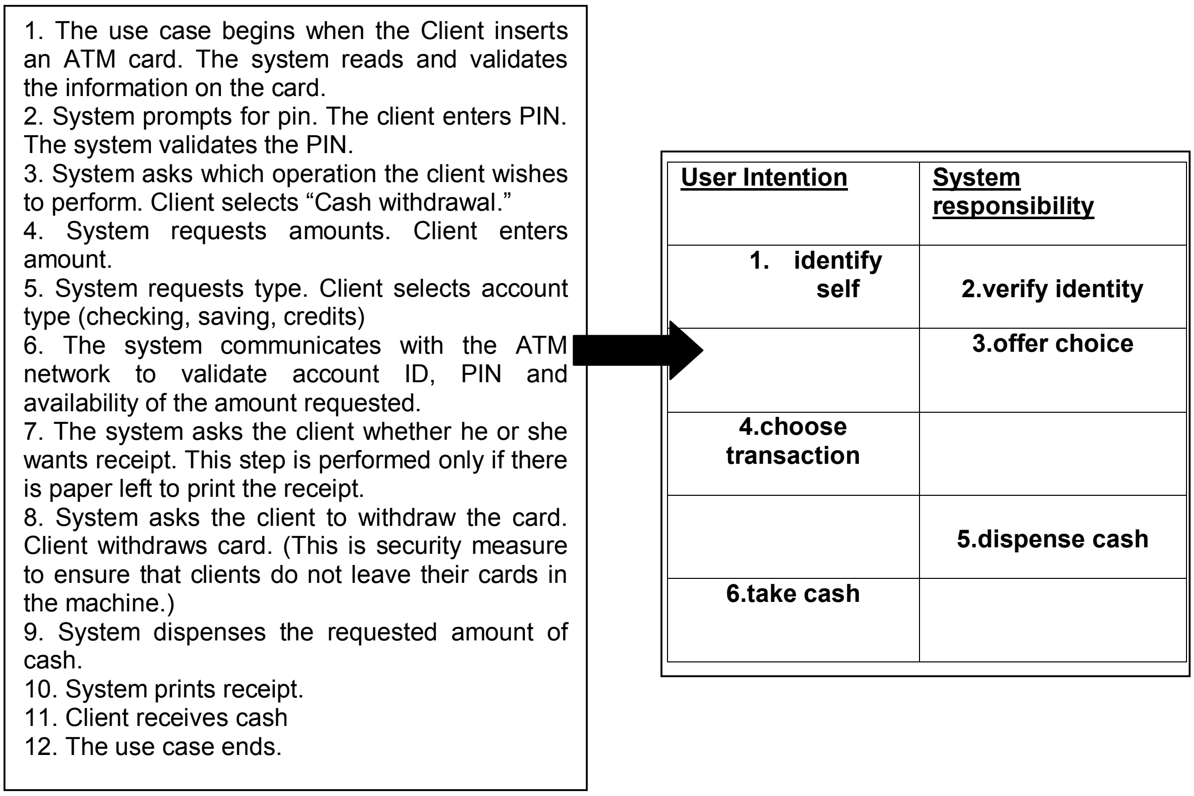
User Intention	System Responsibility
Student identifies himself	Verifies eligibility to enroll via *BR129 Det Eligibility to Enroll.*
	Indicate available seminars
Choose seminar	Validate choice via *BR130 Determine Stu Eligibility to Enroll in a Seminar.*
	Validate schedule fit via *BR143 Validate Seminar Schedule*
	Calculates fees via *BR 180 Calculate Stu* and *BR45 Calculate Taxes for Seminar.*
	Summarize fees
	Request confirmation
Confirm enrollment	Enroll student in seminar
	Add fees to student bill
	Provide confirmation of enrollment

Figure 7. 1: Example of EUI prototype iterates from Essential Use Cases (EUC model (Ambler [10, 11])

7.3 Applying EUI Rapid Prototyping: A Study

To obtain a more rigorous understanding of the reported anecdotal difficulty of using low-fidelity prototyping [71], and EUI prototyping in particular, we conducted a user study of several REs modelling an EUI prototype from a set of requirements written in the form of a user scenario.
Our study participants were 20 post-graduate software engineering students, several of whom had previously worked in the industry as developers and/or REs. All were familiar with requirements and prototyping but none with the EUI prototyping approach. Each participant was given a brief tutorial on the approach and examples of natural language requirements with derived EUC models and EUI prototypes. Participants were then asked to develop an EUI prototype model from an EUC model and natural language requirements. We tracked the time they took to complete the task. The particular

scenario we gave them to analyse was “getting cash”, which is the same scenario discussed in Chapter 3, but here we focus on EUI modelling, given an existing EUC, rather than EUC modelling itself. This scenario is given as we wanted the participants to understand and be comfortable with the requirements before capturing the intended content and the organisation of the UI. The example of the scenario and EUC diagram given are shown in Figure 7.2.



1. The use case begins when the Client inserts an ATM card. The system reads and validates the information on the card.
2. System prompts for pin. The client enters PIN. The system validates the PIN.
3. System asks which operation the client wishes to perform. Client selects “Cash withdrawal.”
4. System requests amounts. Client enters amount.
5. System requests type. Client selects account type (checking, saving, credits)
6. The system communicates with the ATM network to validate account ID, PIN and availability of the amount requested.
7. The system asks the client whether he or she wants receipt. This step is performed only if there is paper left to print the receipt.
8. System asks the client to withdraw the card. Client withdraws card. (This is security measure to ensure that clients do not leave their cards in the machine.)
9. System dispenses the requested amount of cash.
10. System prints receipt.
11. Client receives cash
12. The use case ends.

User Intention	**System responsibility**
1. identify self	**2.verify identity**
	3.offer choice
4.choose transaction	
	5.dispense cash
6.take cash	

Figure 7. 2: Example of Scenario “getting cash” and its EUC diagram

Table 7.1 summarises the results of the study. The correctness (Y correct, X incorrect) and the time taken were recorded for each participant and each EUI component. A correct answer (Y) means that the participant’s component was the same or very similar to an oracle EUI pattern we developed based on the Constantine and Lockwood [4] methodology. Thus:

- Only 42% of the EUI components were correct.
- No participant had all EUI components correct (all Ys). Three had only one wrong (orange highlight). One participant’s components were all incorrect (grey highlight).

- The average time taken was 14.4 minutes, the longest time 35 minutes (yellow highlight) and the shortest 3 minutes. Thus, there was a significant variability in the time taken.

Based on these results, participants were more likely to generate incorrect EUI prototype models than correct ones. The root cause was that participants tended to incorrectly determine the main UI component of a specific business use case. Almost all participants tended to capture unnecessary UI components, gearing towards a concrete GUI rather than EUI components. There was considerable variation in the time taken and the longest time taken did not increase the likelihood of the correctness of the answer: 35 minutes for only 2 correct EUI components. Our survey thus supports the anecdotal findings reported in [71] regarding the problems faced in using low-fidelity prototypes but with more quantitative evidence.

Participant	Answers												Time taken (minutes)
	ID		Other personal detail		Display ID		List of Option		List of Payment		Display Cash		
1	Y			X	Y			X		X		X	7
2	Y		Y		Y		Y			X	Y		10
3	Y			X		X		X	Y			X	5
4	Y			X		X		X		X		X	30
5		X		X		X		X	Y		Y		23
6		X		X		X		X		X		X	10
7	Y			X		X	Y			X	Y		3
8	Y			X		X		X		X	Y		4
9		X		X	Y			X	Y		Y		11
10	Y		Y		Y			X	Y		Y		11
11	Y			X	Y		Y		Y		Y		30
12		X		X	Y			X		X	Y		30
13	Y		Y			X	Y		Y		Y		8
14		X		X		X		X	Y		Y		4
15	Y			X		X		X		X		X	5
16		X		X		X		X		X		X	12
17		X		X	Y		Y			X		X	25
18		X		X	Y			X	Y			X	20
19	Y			X	Y			X	Y		Y		4
20	Y			X	Y			X		X		X	35
	12	8	3	17	10	10	5	15	9	11	11	9	287
Average time taken: 287/ 20= 14.35													

Table 7. 1: EUI prototype Modelling Study Results

7.4 Related Work

Earlier in this chapter we suggested that rapid prototyping is useful in assisting the requirements validation and confirmation as well as supporting the analysis and requirements engineering at an early stage. Many other researchers have generated user interfaces or prototypes in the requirement engineering domain.

For example, Ogata and Matsuura propose a method for automatic generation of user interface/ prototypes for web-based business applications based on requirement specifications defined in UML [142]. Their work guarantees consistency of the data and flow between the requirement Analysis model and prototype, and thus decreases the time taken to conduct requirements analysis.

Gabrysiak et al. present an approach and preliminary tools that support combining requirement models with specific requirements prototyping of interactive visualisations (animations). These are for requirements elicitation and validation of systems for multiple users in the business domain and for scenario-based requirements [151].

Furthermore, Li et al. present an approach for validating system requirement at an early stage by transforming UML system requirement models with Object Constraint Language(OCL) specifications into executable prototypes with the aim of checking multiplicity and invariant constraints [152]. Their work also performs automatic consistency checking of requirements. However, some OCL expressions cannot be handled by their tools and the algorithm used does not support larger executable sets of OCL [152].

Memmel and Reiterer introduce an interactive integration between interdisciplinary and informal modeling language, comprising different fidelity levels for UI prototyping in their INSPECTOR tool [149]. This enhances the traceability of dependencies, increases transparency of design decisions and provides support for round-trip engineering [149].

Schneider developed the “Fast feedback” technique with “By Product principles” to collect additional information during interviews with clients by sketching and animating user interface mock–ups guided by use case steps [25]. This technique requires two interviewers to collect requirements: one interacts with the stakeholder and the other completes a template.

There is also much work on non tool-based techniques. Vijayan and Raju propose a paper prototyping approach for eliciting requirements [153]. The requirements gathered are validated by examining the captured paper prototype to identify omissions, ambiguities and other requirement quality problems[153].

Molina et al. have developed a model and graphical notation for the specification of abstract UIs based on a conceptual pattern [154]. This Just UI approach identifies patterns for UIs and abstracts them to work with problem domains specifically for presentation and navigation issues. It extends Object- Oriented(OO) methods to capture UI requirements and presents a set of patterns that can be used as building blocks to create UI specifications for information systems manually [154].

The work discussed above present approaches to generate the user interface/ prototype for requirement engineering purposes, mainly for requirement analysis. However, most approaches generate the user interface/prototype from semi-formal specifications, typically UML models, only and not from informal specifications in the form of textual natural language requirements. Some of the work identified lacks automation support to generate the interface/prototype and to conduct the validation process. None of the approaches generate EUI prototypes; most generate another type of abstract prototype to visualise the requirements. Almost no work has been done to develop a EUI pattern library or EU pattern template for generating the user interface/prototype. In addition, most of the approaches also lack round-trip prototyping support to validate the requirements by both the REs and clients.

7.5 Our Approach

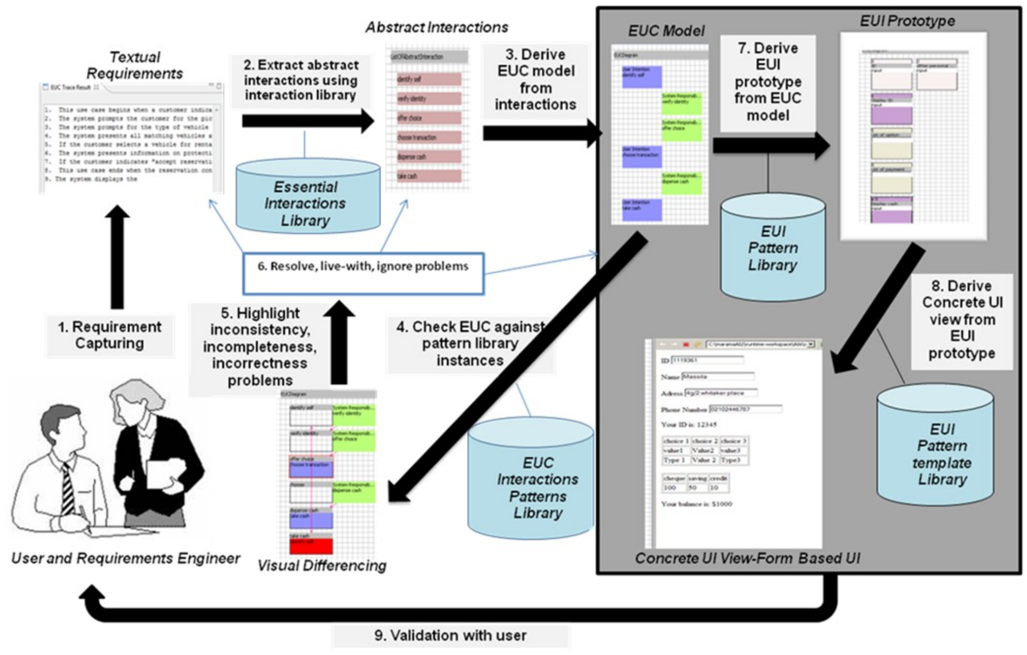

Textual Requirements
2. Extract abstract interactions using interaction library
Abstract Interactions
3. Derive EUC model from interactions
EUC Model
7. Derive EUI prototype from EUC model
EUI Prototype
Essential Interactions Library
EUI Pattern Library
6. Resolve, live-with, ignore problems
8. Derive Concrete UI view from EUI prototype
1. Requirement Capturing
5. Highlight inconsistency, incompleteness, incorrectness problems
4. Check EUC against pattern library instances
EUC Interactions Patterns Library
EUI Pattern template Library
User and Requirements Engineer
Visual Differencing
Concrete UI View-Form Based UI
9. Validation with user

Figure 7. 3: End-to-end EUC and EUI prototyping approach

We were motivated by the gaps that we found from the related work discussed in the previous section. We were also surprised by the results in section 7.3, although perhaps should not have been, given the results from per manual EUC extraction study presented in Chapter 4. Many participants were experienced in the field of software requirements. Given this background, we expected that less time would be used to accomplish the simple task. We also expected they would be able to develop more accurate EUI rapid prototypes for this problem domain.

This has provided us with an additional motivation to develop an approach and supporting tool to enable REs to capture or confirm more effectively requirements with clients via end-to-end rapid prototyping using low-fidelity EUI prototyping together with a concrete UI prototype. Figure 7.3 provides an overview of our end-to-end rapid prototyping and requirements elicitation process.

I. The process followed in items (1-6) in Figure 7.3 is as per described in earlier chapters, Chapter 4, 5, and 6. Our new work presented here (in the grey box) allows the RE to automatically and traceably transform EUC models to EUI prototypes using a novel EUI pattern library we have developed (7).

II. Combined with our earlier toolset, this means traceability is provided throughout the process, allowing any of the EUI components to be traced forward/back from/to the EUC model, abstract interaction or textual natural language requirement.

III. The EUI prototype can also be translated to a more concrete form-based UI view, an HTML form, by using a novel EUI Pattern template library (8). An EUI prototype model can be translated to a concrete form-based UI using a pre-defined template in a EUI pattern template library, one template for each EUI pattern. The EUI Pattern template consists of the descriptions of Concrete UI components to be instantiated for a particular EUI pattern.

IV. Simple interaction with the generated HTML form is supported to illustrate how target system information input and output could work. This EUI model and concrete UI can then be reviewed by the RE with end-users to validate and confirm the consistency of the original textual requirements (9).

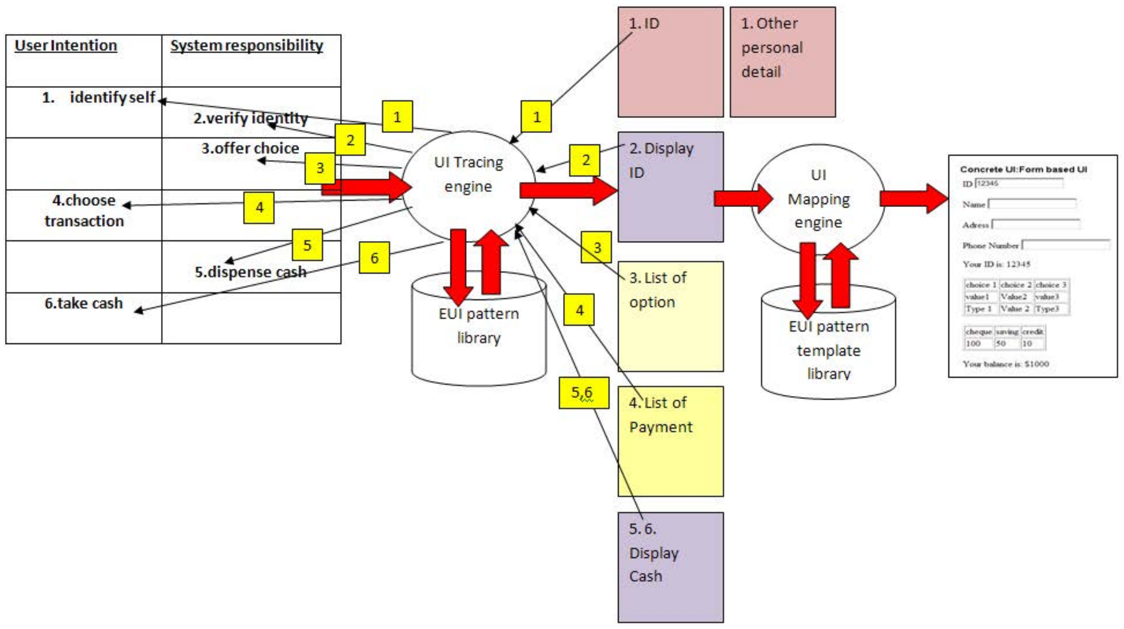


Figure 7. 4: An example of performing mapping of EUC model to EUI prototype using the UI Pattern library with trace-forward/ trace-back and translating the EUI prototype to the concrete UI-HTML form

Figure 7.4 shows in more detail the mapping and tracing process between the EUC model, EUI prototype and Concrete UI view, using the "getting cash" scenario. The numbers indicate mapped elements between the models. The EUC model is mapped to/from the EUI prototype using the UI mapping engine. This maps each of the abstract interaction components which have a relevant EUI pattern in the EUI pattern library. For example, the abstract interaction “identify self” (1) will be searched for in the EUI pattern library and its related EUI pattern found. This results in abstract UI elements “ID” and “Other personal detail” being added to the EUI model. More than one abstract interaction may share the same EUI pattern. For example, the abstract interactions “dispense cash” and “take cash” share the same UI pattern “Display cash”. Here, only one EUI pattern “Display Cash” is included in the model with two different sequence numbers associated. The sequence associations support trace forward and back.

7.6 EUI Pattern Library

We developed the EUI patterns in the EUI Pattern library, using an adaptation of the brainstorming methodology proposed by Constantine and Lockwood [4]. The adaptation generalised the approach providing a simpler and more generic EUI pattern for EUI prototypes. The generalised EUI pattern comprises four types of EUI pattern category: List, Display, Input and Action. These are similar to the concept of Containers, introduced by Constantine and Lockwood. The main aim of these EUI Patterns is to assist REs to rapidly model a user interface based on the requirements captured and modelled earlier in the EUC model. An abstract UI captured using such a pattern is used as a medium for early communication between the RE and the client as it is easy to understand and allows the client to narrow down UI detail before moving to the concrete UI.

Table 7.2 shows examples of mappings among abstract EUC interactions (right) and various EUI patterns (centre), and their categories (left). For example, the EUI pattern “Save” from the “Action” category is associated with three different abstract EUC interactions: “record call”, “record detail” and “save identification”. We can see that the abstract EUI patterns are very general and apply across a range of different domains. For example, the EUI pattern “Save” could support a range of different scenario domains such as making calls in a mobile application domain to online booking, registration and retail systems.

In more detail, the four EUI pattern categories are as follows.

- **List**: Show a list of items, options or values that are associated with a particular abstract interaction of the EUC model. Default values are provided from the UI pattern library but can be overridden during application.
- **Display**: Display output based on an associated abstract interaction of the EUC model. This could display a name, id, number, address, message or notification.

- **Input**: Allow a user to input data or details of a specific element associated with an abstract EUC interaction.
- **Action**: Show a control button, such as save, delete and submit, based on an associated EUC abstract interaction.

EUI pattern category	EUI pattern	Abstract interaction
List	List of option	Choose
		offer choice
		Select option
	List of solution	offer alternative
		offer possible solution
	List of product	select product
	List of problem	view problem
	List of payment	choose transaction
		choose payment
		select amount
Display	Display payment	validate payment
		show payment
	Display Item detail	return item
		view detail
	Display status	check user
		Notify user
	Display ID	verify identity
		provide identification
	Display error message	display error
Input	ID	identify self
		request identification
	Other personal detail	identify self
		request identification
	Payment detail	make payment
	Item detail	provides detail
	Number	make call
		indicates number to dial
Action	Help	Ask help
		Present solution
	Save	record call
		Record detail
		save identification
	Print	Print
	Delete	delete item
	Submit	insert information

Table 7. 2: Example of EUI pattern Category and its related EUI pattern and it's associated Abstract Interaction from the EUC model

7.7 EUI Pattern Template library

The EUI Pattern template library consists of an EUI Pattern template which is developed to translate the EUI prototype to the concrete UIs in a form of HTML page. The EUI pattern template is based on the EUI pattern used in the EUI prototype. The EUI pattern template is already pre-defined in the library. It contains templates defined in HTML format for each of the EUI pattern categories: List, Display Input and Action. The defined EUI Pattern template for the HTML form is as below;

i. List: Table
ii. Display: message/text/data/value
iii. Input: Text Input
iv. Action: Button

The EUI pattern template is also applicable and reusable for various domains of applications. Table 7.3 shows the examples of EUI pattern templates with their associated EUI patterns and domains applicable to the pattern.

EUI pattern categories	EUI Pattern	EUI Pattern template	Domains
Action	Submit	Button	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	Add		
	Search		
List	List of item	Table	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	List of payment		
	List of option		
Display	Display availability	Numbers/text	Online banking, online booking, online business, e-commerce, online voting system, mobile system, online reservation
	Display amount	Value/text	
	Display ID	Numbers	
Input	Item detail	Text input	Online banking, online booking, online business, e-commerce, online voting system, online reservation
	Payment detail		
	Problem form		

Table 7. 3: Examples of EUI Pattern template with its associated EUI Pattern and associated Domains in the EUI Pattern template library

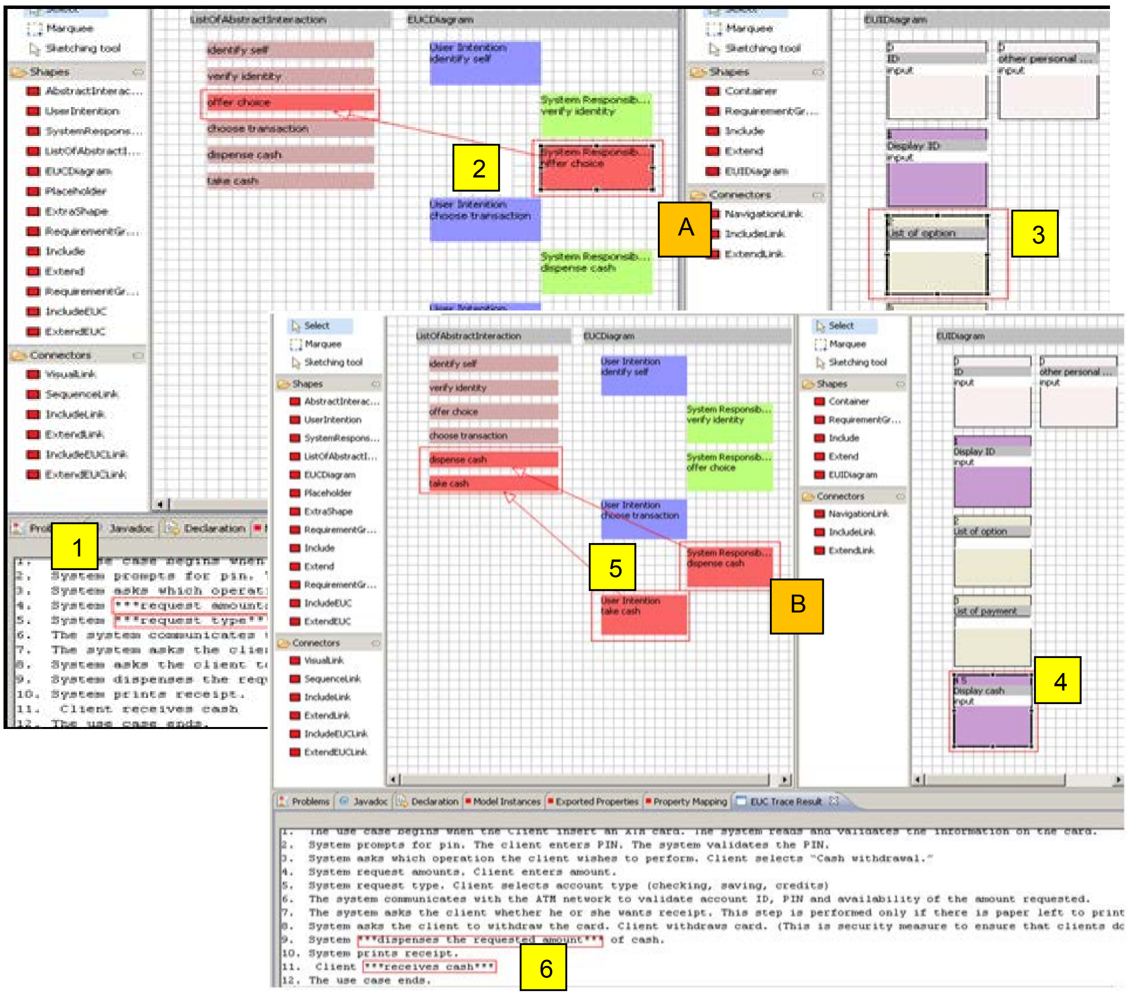
.

7.8 Tool Support

We have extended our prototype tool, MaramaAI, which previously supported extraction of EUC models from textual natural language requirements, as described in previous chapters, to additionally and automatically map EUC models to EUI prototypes and concrete UIs, based on the approach outlined in the previous section. The EUI prototype is modelled in a Marama editor called MaramaEUI. The concrete UI is presented in the form of an HTML page, both realised in the Eclipse IDE.

Several screen dumps of the tool in use are shown in Figure 7.5. From a set of textual natural language requirements (1);

I. semi-formal EUC models are extracted (2) and
II. then mapped to a low-fidelity Essential User interface model in a MaramaEUI editor (3).



```
1.  The use case begins when the client insert an ATM card. The system reads and validates the information on the card.
2.  System prompts for pin. The client enters PIN. The system validates the PIN.
3.  System asks which operation the client wishes to perform. Client selects "Cash withdrawal."
4.  System request amounts. Client enters amount.
5.  System request type. Client selects account type (checking, saving, credits)
6.  The system communicates with the ATM network to validate account ID, PIN and availability of the amount requested.
7.  The system asks the client whether he or she wants receipt. This step is performed only if there is paper left to print
8.  System asks the client to withdraw the card. Client withdraws card. (This is security measure to ensure that clients dc
9.  System ***dispenses the requested amount*** of cash.
10. System prints receipt.
11.  Client ***receives cash***
12. The use case ends.
```

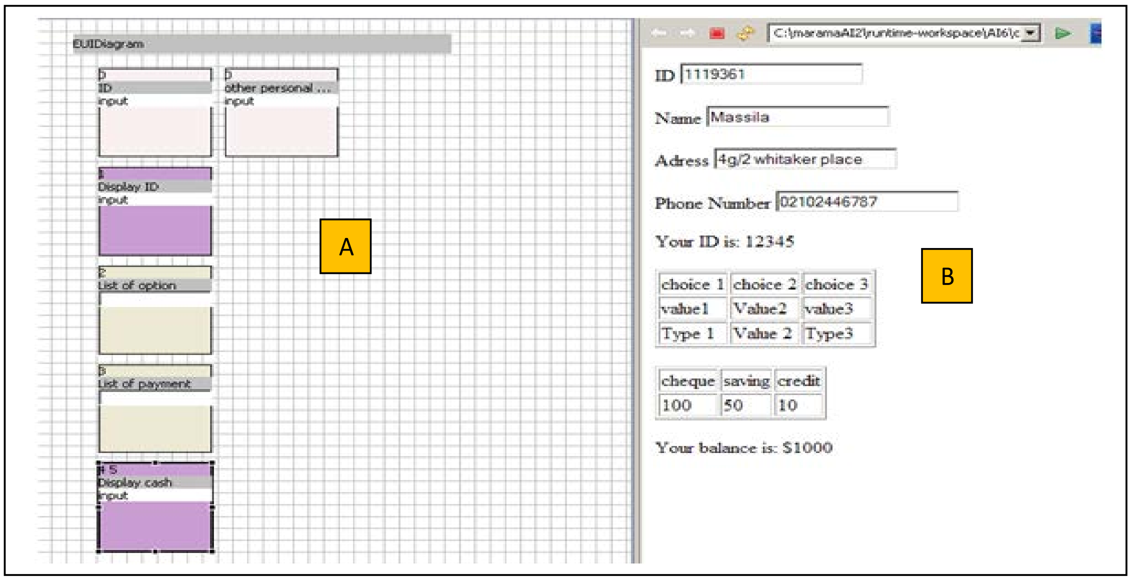
Figure 7. 5: Trace forward and Trace-back from EUC model to EUI prototype.

Each EUI prototype component is associated with an EUC model abstract interaction component and through that, the original textual natural language requirements from which it was derived from. Any EUI component can be selected and its associated EUC component and related textual natural requirements can be shown using a “trace back” menu item which highlights the relevant components. Here, the tool differentiates the EUI pattern categories with different colours which follow the concept of the EUI component by Constantine and Lockwood [4]. The List category is visualised in light yellow, the Display category in purple, Input category in light pink and the Action category in blue.

For example in Figure 7.5 (top – section A), the Option list EUI Component (3) is traced back to a “system responsibility: offer choice” EUC component (2) which in turn is traced back to the textual requirement (1), both of which have been highlighted. One EUI component might be associated with more than one abstract interaction in the EUC model. Figure 7.5 (bottom - section B) shows that the

EUI component " Display cash" (4), traces back to two abstract interaction components of the EUC model "dispense cash" and "take cash" (5) and the associated textual requirement (6) "dispenses the requested amount" and "receives cash".

The idea of this support is for REs to confirm that the requirements captured or described earlier in textual natural language and the EUC model are consistent with the client's original requirements. Further, the RE could use this automated support to obtain fast feedback from clients for the captured or gathered requirements based on their understanding. This will allow any inconsistency to be detected as early as possible. It also shows that the EUI pattern is only abstracting the main important components that need to be in the user interface in order to display the core requirements captured by the EUC model. It does not display unimportant components of the system and does not justify any technology options for the system. This is because at the early stage of requirements analysis, neither fancy, colourful layout of user interface nor technology-dependent identification is required. The focus is to understand the problem first [25] and to confirm the consistency of the requirements from both RE and client perspectives, especially in terms of behavioural or functional requirement. The RE therefore has the flexibility to agree or disagree with the results provided by the tool.



ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

Your ID is: 12345

choice 1	choice 2	choice 3
value1	Value2	value3
Type 1	Value 2	Type3

cheque	saving	credit
100	50	10

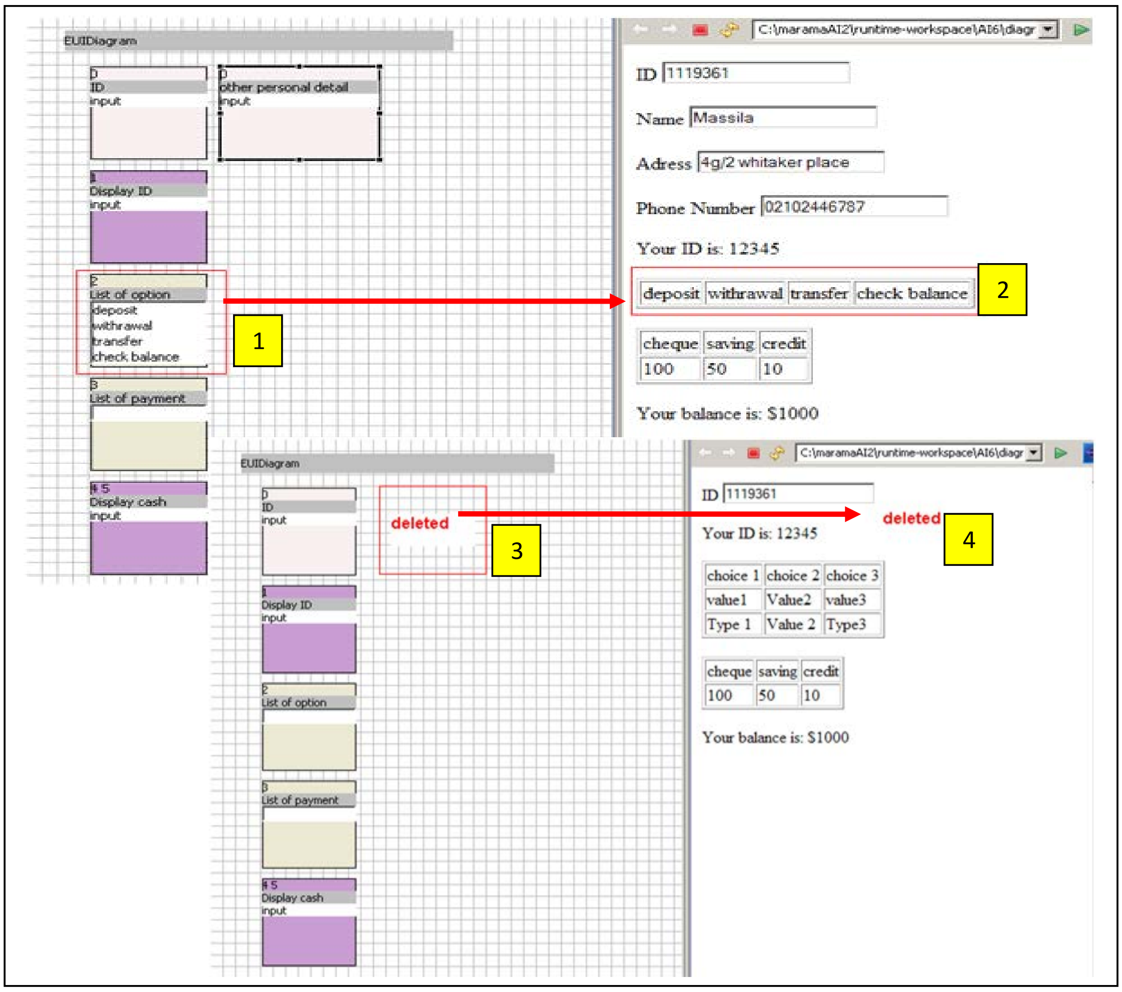
Your balance is: $1000

Figure 7. 6: Marama EUI and concrete UI view in a form of form- based UI

Figure 7.6 shows the prototype view for both the Marama EUI (A) and concrete HTML form-based UI view (B). Both views allow the RE and client to walk-through the requirement and its UI in order to validate the consistency of the requirement. The Marama EUI component can be edited, allowing the

RE and client to add input detail that they think is required for the UI, or to delete any EUI pattern that they think is not necessary for a specific business use case.

The concrete UI view helps clients with a non-technical background to understand the whole process and to confirm at an early stage that the requirements captured by the RE are consistent with their original needs, before the requirement is passed to a developer or designer.



ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

Your ID is: 12345

deposit	withrawal	transfer	check balance

cheque	saving	credit
100	50	10

Your balance is: $1000

ID 1119361

Your ID is: 12345

choice 1	choice 2	choice 3
value1	Value2	value3
Type 1	Value 2	Type3

cheque	saving	credit
100	50	10

Your balance is: $1000

Figure 7. 7: Modification of EUI prototype - Addition and Deletion in EUI prototype

Figure 7.7 shows a modification of an EUI prototype by adding a new detail to the “List of option” EUI pattern (1) and the result of the modification in the concrete UI view (2). Here, the “List of option” in (1) is extended with more detail: “deposit, withdrawal. Transfer and check balance”. This causes a change in the concrete UI view by displaying the additions in (2). In (3), one of the UI patterns, “Other personal detail", has been deleted resulting in changes to the concrete view (4), where the UI elements requesting "name, address, phone number" have been deleted.

7.9 Architecture and Implementation

We developed the MaramaEUI toolset using our Marama meta-tools [120] and a number of specialised components: extraction from the textual natural language requirements (described earlier), EUI model mapping from the EUC model, HTML form generation from the EUI model, extended tracing support between the EUI models, EUC model, abstract interactions and the textual natural language requirements, model consistency support and visual differencing. An outline of the tool's architecture is represented in Figure 7.8.

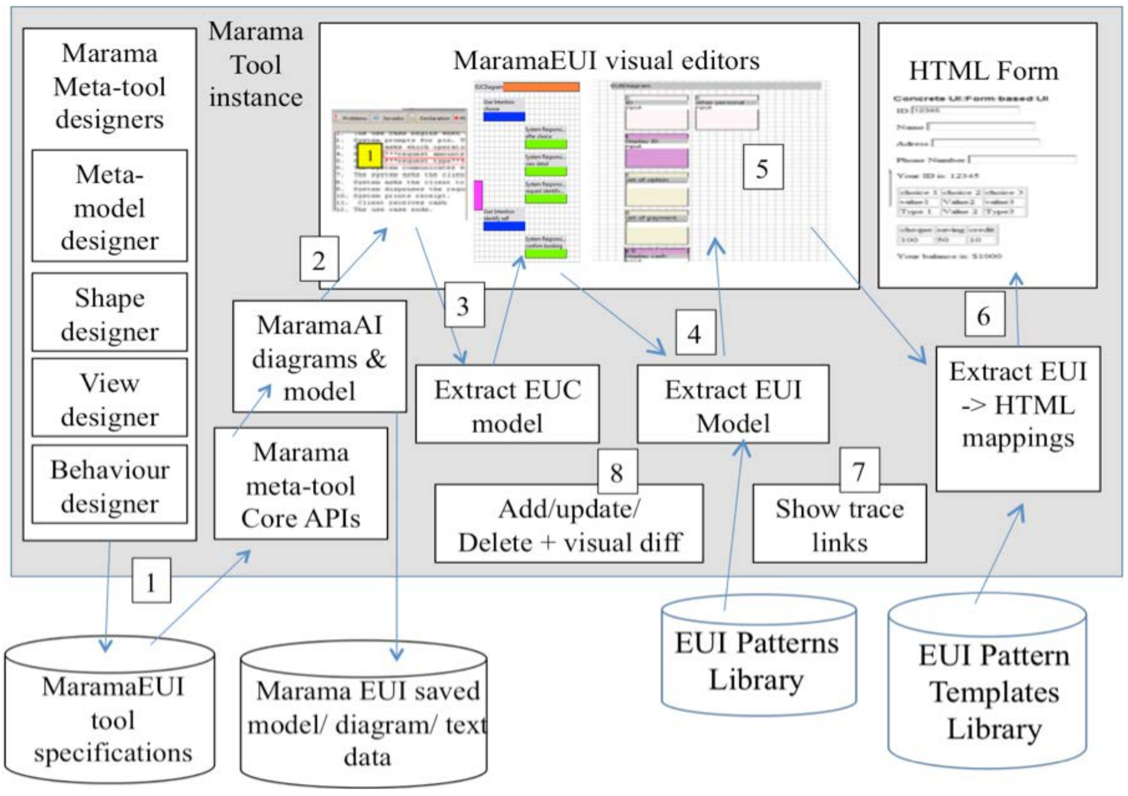

Marama Meta-tool designers
Meta-model designer
Shape designer
View designer
Behaviour designer
Marama Tool instance
MaramaEUI visual editors
HTML Form
MaramaAI diagrams & model
Marama meta-tool Core APIs
Extract EUC model
Extract EUI Model
Extract EUI -> HTML mappings
Add/update/ Delete + visual diff
Show trace links
MaramaEUI tool specifications
Marama EUI saved model/ diagram/ text data
EUI Patterns Library
EUI Pattern Templates Library
1
2
3
4
5
6
7
8

Figure 7. 8: MaramaEUI tool architecture.

i. Items (1-3) are as described in Chapter 5 and 6. We then map an EUC model into an EUI model using our EUI patterns library (4), as described in Section 7.5.

ii. The generated EUI model may be edited by the RE to add, update or delete automatically populated items, to rearrange the interface and to annotate the interface (5).

iii. Multiple EUI models can be generated from multiple EUC models. A HTML form can be generated from each EUI model using our EUI pattern template library (6), that maps EUI elements and relationships into HTML form elements.

iv. The form can interact with Eclipse in limited ways to illustrate the likely system interface characteristics to stakeholders. Tracing support is provided between the EUI models, EUC model, abstract interaction and textual natural language requirement (7), where a selected EUI item will have derivative EUC item(s), abstract interaction and the textual natural language requirements highlighted. This is potentially a many-to-many mapping.

v. Model consistency is maintained between a generated EUI model with the originating EUC model, as described and illustrated in (8). Adding, updating and deleting items in either model are propagated to the other and a visual differencing approach uses annotation to highlight affected items in other models.

   We used the Marama meta-tool which supports rapid design and development of domain-specific visual languages to develop the notations and editors for the EUC and EUI models, the EUI to EUC tracing highlighting, and the visual difference of changes to highlight changes made to EUI and EUC diagrams. The meta-model and DSVL editors were supplemented with event handlers to provide low-level model constraints, consistency management support and interfaces with other elements of the architecture. These were implemented in Java and include generation of dialogues and problem markers to assist the user to track and resolve inconsistencies.

vi. Further event handlers were implemented in Java to implement generation of HTML forms from EUI models, to generate an EUI model from an EUC model and to trace back the EUI model to other requirement components: EUC model, abstract interaction and textual natural language requirement were also used. The event handlers used in this work are as follows.

    i. Map EUC to EUI

       The event handler for mapping the EUC model to MaramaEUI, “Map EUC to EUI”, helps to generate the EUI prototype automatically. The event handler works with the mapping engine to map the EUC model to an EUI prototype. The mapping engine analyses and matches the abstract interaction of the EUC model with the property in the EUI Pattern library. Then, the abstract interaction of the EUC model is mapped automatically to the EUI prototype based on its EUI pattern category. The event handler will not map the newly-added EUC component to the EUI prototype if the abstract interaction does not exist in the EUI Pattern library. A sequence diagram to illustrate the interaction is shown in Figure 7.9.

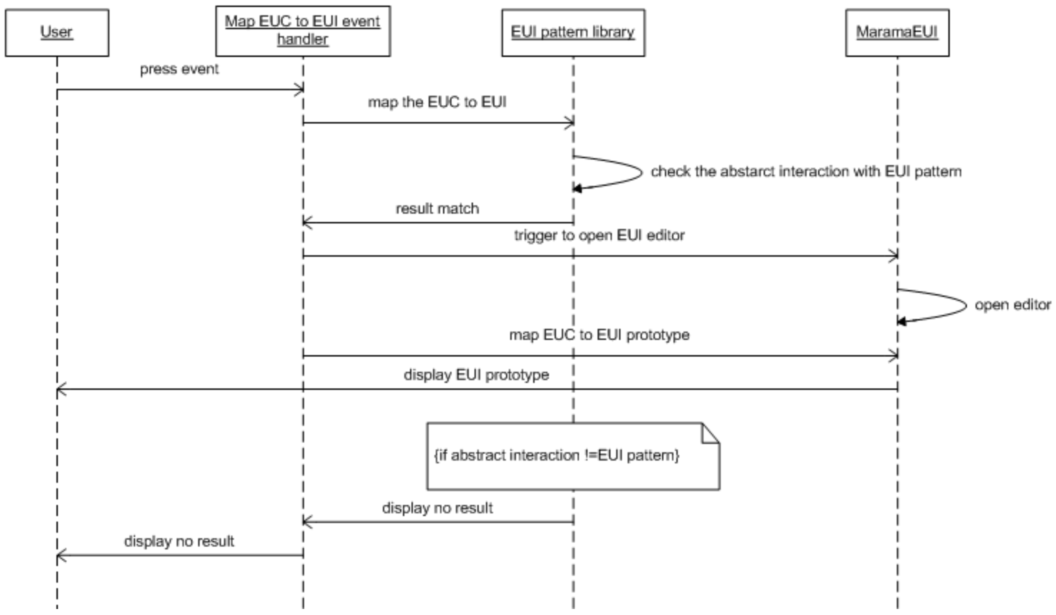


Figure 7. 9: Example of Map EUC to EUI

ii. Trace Back

To trace back the EUI prototype component to its source, we used the help of the Trace Back event handler. This event handler also works together with the tracing engine. The selected EUI prototype component is analysed by the tracing engine and then matched with the abstract interaction in the EUI Pattern library. If we try to trace back the EUI component, the tool will show where the associated abstract interaction, EUC model and essential interaction for that particular EUI prototype come from. If a newly added component of the EUI prototype does not match with the abstract interaction in the EUI Pattern library, no result is provided. A sequence diagram to illustrate the interaction is shown in Figure 7.10.

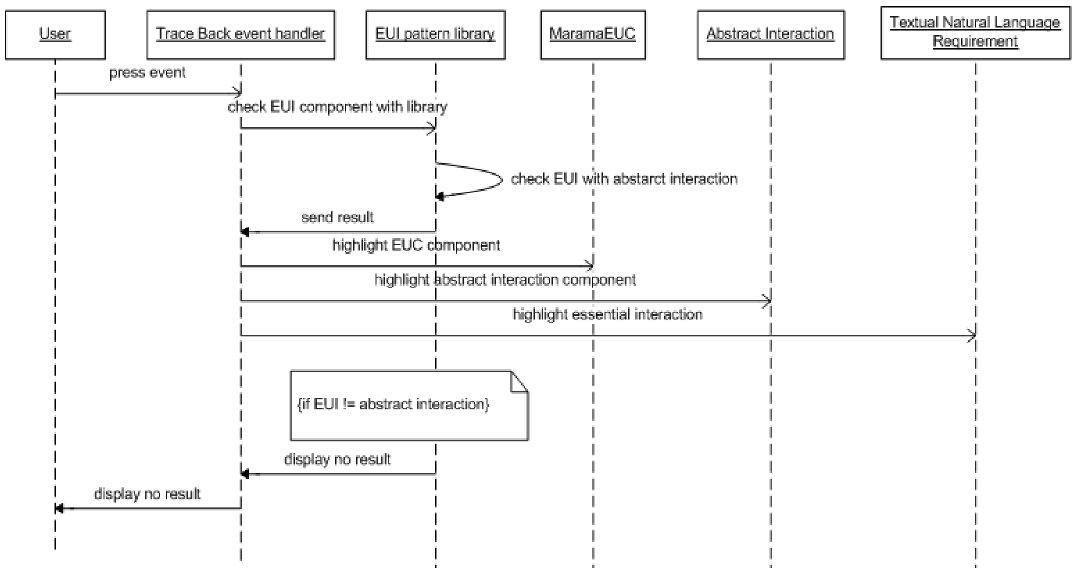

User
Trace Back event handler
EUI pattern library
MaramaEUC
Abstract Interaction
Textual Natural Language Requirement
press event
check EUI component with library
check EUI with abstarct interaction
send result
highlight EUC component
highlight abstract interaction component
highlight essential interaction
{if EUI != abstract interaction}
display no result
display no result

Figure 7. 10: Example Trace Back from EUI prototype to EUC Model

iii. EUI to prototype

The event handler for translating the EUI prototype to concrete UI view-HTML form called “EUI to prototype”, helps to generate the concrete UI view in a form of HTM form automatically. The event handler works with the mapping engine to map the EUI prototype to HTML form. The mapping engine analyses and matches the EUI pattern of the EUI prototype with the property in the EUI Pattern template library. Then the EUI pattern of the EUI prototype is mapped automatically to the HTML form based on the matching EUI pattern template. The event handler will not map the newly added EUI prototype component to the HTML form prototype if the UI pattern does not exist in the EUI Pattern template library. A sequence diagram to illustrate the interaction is shown in Figure 7.11.

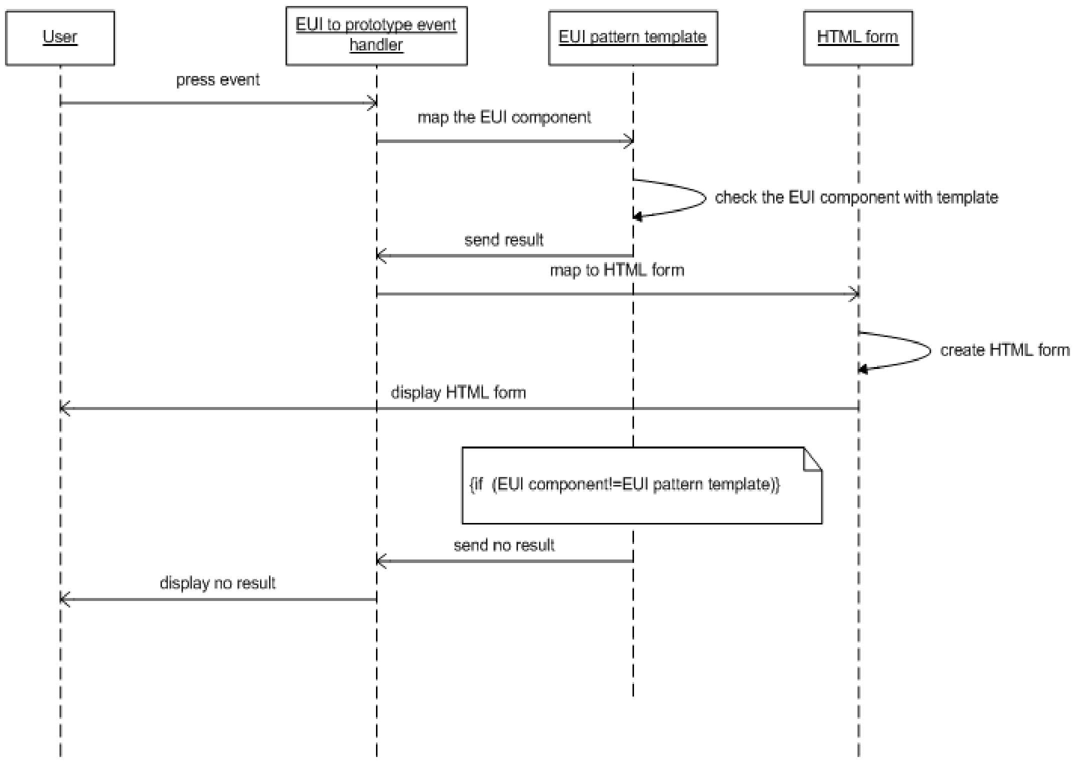


Figure 7. 11: Example of Generating HTML form from the EUI prototype

7.10 Evaluation

In our preliminary study, we demonstrated that end users find manual derivation of EUI prototypes difficult, time consuming and error prone. We wanted to demonstrate the effectiveness of our new automated tool support in Marama EUI, together with its ability to support end-to-end prototyping. To this end we conducted an end user study to evaluate user perceptions of the tool and its application.

Participants in this study were the 20 software engineering post-graduate students who had earlier participated in the manual study of modelling EUI prototype. The same scenario as the manual study is used. Here, their experience as REs could be categorised as novice to intermediate. Each participant was given a brief tutorial on how to use the tool and some examples of how the tool captures requirements using end-to-end prototyping. They then derived an EUI prototype from an EUC model and natural language requirements and mapped the EUI prototype to a concrete UI view. Further exercises modifying the EUI prototype followed: adding and deleting EUI components and exploring the result of the modifications in the concrete UI view.

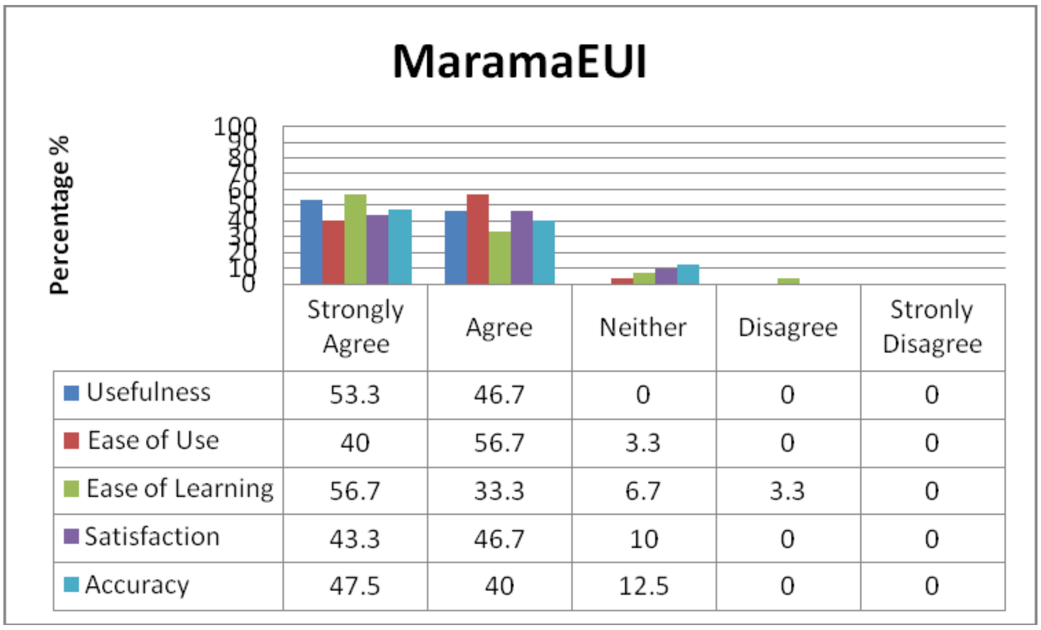
Having familiarised themselves with the tool's capabilities and undertaken the tasks, users completed two surveys:

1. MaramaEUI itself and
2. Formal evaluation of end-to-end prototyping.

However, in this chapter, we only discuss the result of the first survey. The second survey results are discussed in the Chapter 9. This chapter focuses only on Marama EUI and comprises a standard evaluation by Lund [156] of user perceptions of the usefulness, ease of use, ease of learning, satisfaction and accuracy of the EUI patterns in supporting EUI prototyping. The first survey consisted of two questions for accuracy, four questions for usefulness and three for other characteristics. A five-part Likert scale was used for each question. The type of questions for each characteristic is in Table 7.4.

User Perception Characteristics	Questions
Usefulness	It is useful to capture the abstract prototype.
	It helps me be more effective in capturing Essential User Interface prototype (EUI).
	It makes me easier to understand requirements that has been modelled earlier using EUC.
	It makes me easier to confirm the requirements with the client from the early stage.
Ease of Use	It is easy to use.
	It is user friendly.
	I don't notice any inconsistencies as I use the tool.
Ease of Learning	I learned to use it quickly.
	I easily remember how to use it.
	It is easy to learn to use it.
Satisfaction	I am satisfied with it.
	I would recommend it to a friend.
	It is fun to use.
Accuracy	I think the EUI pattern provided for the EUI prototype is accurate as what I expected.
	I think the abstract interaction component of the Essential Use Cases model is mapped accurately to the EUI prototype

Table 7. 4: User Perception Characteristics and Questions Evaluating Them



	Strongly Agree	Agree	Neither	Disagree	Stronly Disagree
🟦 Usefulness	53.3	46.7	0	0	0
🟥 Ease of Use	40	56.7	3.3	0	0
🟩 Ease of Learning	56.7	33.3	6.7	3.3	0
🟪 Satisfaction	43.3	46.7	10	0	0
🟦 Accuracy	47.5	40	12.5	0	0

Figure 7. 12.: User study results of Marama EUI-Usefulness, ease of use, ease of learning, satisfaction and accuracy

Figure 7.12 shows the results for the standard usability and accuracy results of the EUI pattern survey conducted on Marama EUI alone. For each characteristic, the results of each corresponding four, three and two questions block were averaged to produce the results shown. The results are very positive, with strong agreement over the usefulness of the tool (about 90% strongly agree or agree on its usefulness), the ease of use (over 90%), ease of learning (about 90%), satisfaction (about 90%) and accuracy of the results of EUI patterns provided (over 85%). The few disagreements over ease of learning related to a preference by those participants to have a video demo embedded in the tool to assist in learning to use it.

Overall, these results are very encouraging, particularly given prior studies, our own and others, that suggest low-fidelity prototype or abstract prototyping, while appealing to end users, have a large barrier to entry due to the effort involved [71]. The accuracy result is also very positive as most participants felt that the EUI patterns we developed help to enhance the accuracy level of the UI components in the EUI prototypes. A minority thought that the tool would be constrained by the coverage of the EUI Pattern library. An automatic updating of the library was suggested to allow other REs or users to update the library automatically rather than just depending on the library provided by the developer. For this, it was also suggested that guidelines for developing EUI patterns to be embedded in the tool.

7.11 Summary

We have described an approach supporting the confirmation and verification of requirements from both RE and client perspective using an end-to-end rapid prototyping. An initial study of the manual usage of EUI prototyping was conducted and the poor results gained confirmed the point of view of Robertson [71] on the real problems faced by the engineers in effectively using low-fidelity prototyping, and subsequently motivated our work.

We have developed an automated tool support for our approach to help overcome the problems faced in manually applying EUI prototyping. We have also evaluated our prototype tool using an end user study for MaramaEUI. The results of this evaluation are promising, with most participants finding our tool to be useful for validating requirements, especially in confirming the consistency. A formal usability survey for the end-to-end prototyping approach provided by our tool was also conducted and the results are discussed in Chapter 9. The generic use of our tool is discussed in the next chapter by showing three examples of case studies from different domains.

In our new approach, we believe that requirements captured earlier by an RE can be verified with the client through the visualisation of low-fidelity prototypes in a form of Essential User Interface prototypes and also in a more concrete form-based UI to validate requirements. While the evaluations described in Chapter 9 do not extend to client participants, just focussing on the REs, including client participants will be a focus of our future work.

Chapter 8: Case Studies Examples

This chapter describes three different case studies of requirements written in a form of user scenarios that we use to demonstrate and describe the key features of our proof of concept tool - MaramaAI (Automated Inconsistency Checker). The key features described are in capturing requirements with Essential Use Cases (EUC), in checking the consistency and validating the requirements and also in supporting the end-to-end rapid prototyping.

8.1 Introduction

Three different case studies of requirements, written in the form of user scenario from different domains of applications, are described and which our toolset are aplied to. This is to demonstrate and describe the key features of our proof of concept tool -MaramaAI. The key MaramaAI tool features illustrated for each scenario are:

a) Capturing the requirements,

b) Checking the consistency and validating the requirements, and

c) Supporting end-to-end rapid prototyping.

We chose three diverse sets of requirements examples. The first set of requirements is a scenario of reserving a vehicle from a rental company, written by Evans [1]. The second is a book check-out scenario of a library system written by Sendall and Strohmeier [2] which illustrates a user scenario with the use of extensions in the description. This allows us to demonstrate our tool support for the use of extend/include. The third requirements provide multiple scenarios of a real industry project example for managing events by Silicon Dream Ltd. This allows us to demonstrate our tool's key features in handling multiple requirements with real industry requirements.

8.2 Case Study 1: Reserve a Vehicle from a Rental Company

We chose this user scenario, which was developed by Evans and published on the IBM developer works website, as a first example of a requirement to demonstrate the key features of our tool. This user scenario is a "hypothetical browser-based software system for an auto rental company" [1] mainly for an individual account. It illustrates the situation that happens in a rental company when a customer comes to the rental counter to rent a vehicle [1]. It is also an example from an online booking domain of application. The description of this user scenario is shown in Figure 8.1.

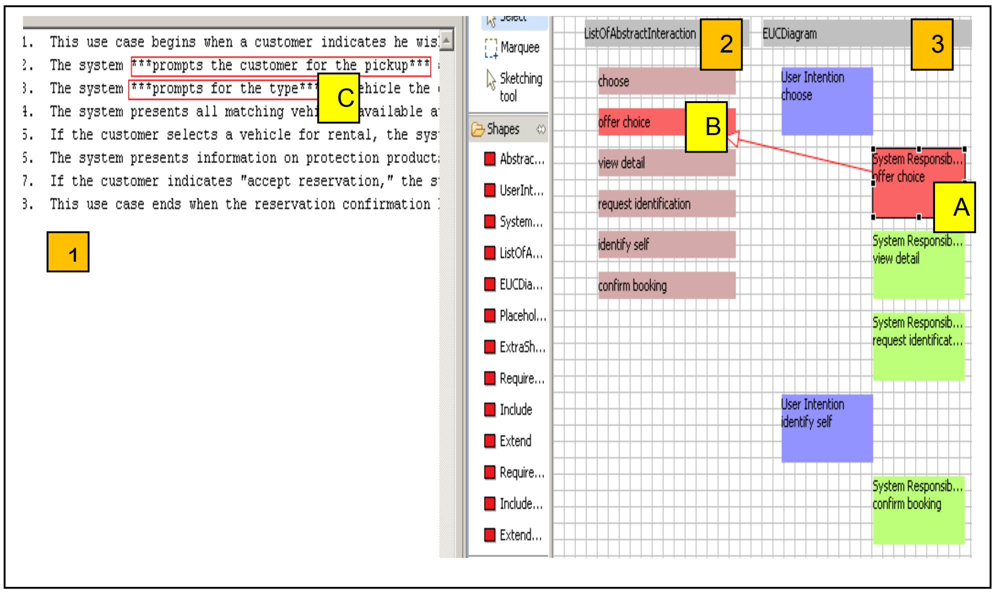
1. This use case begins when a customer indicates he wishes to make a reservation for a rental car.
2. The system prompts the customer for the pickup and returns locations of the reservation, as well as the pickup and return dates and times. The customer indicates the desired locations and dates.
3. The system prompts for the type of vehicle the customer desires. The customer indicates the vehicle type.
4. The system presents all matching vehicles available at the pickup location for the selected date and time. If the customer requests detailed information on a particular vehicle, the system presents this information to the customer.
5. If the customer selects a vehicle for rental, the system prompts for information identifying the customer (full name, telephone number, email address for confirmation, etc.). The customer provides the required information.
6. The system presents information on protection products (such as damage waiver, personal accident insurance) and asks the customer to accept or decline each product. The customer indicates his choices.
7. If the customer indicates "accept reservation," the system informs the customer that the reservation has been completed, and presents the customer a reservation confirmation.
8. This use case ends when the reservation confirmation has been presented to the customer.

Figure 8. 1: Example of User Scenario: Reserve a Vehicle [1]

8.2.1 Example of Usage

We demonstrate our tool’s key features with the user scenario below:

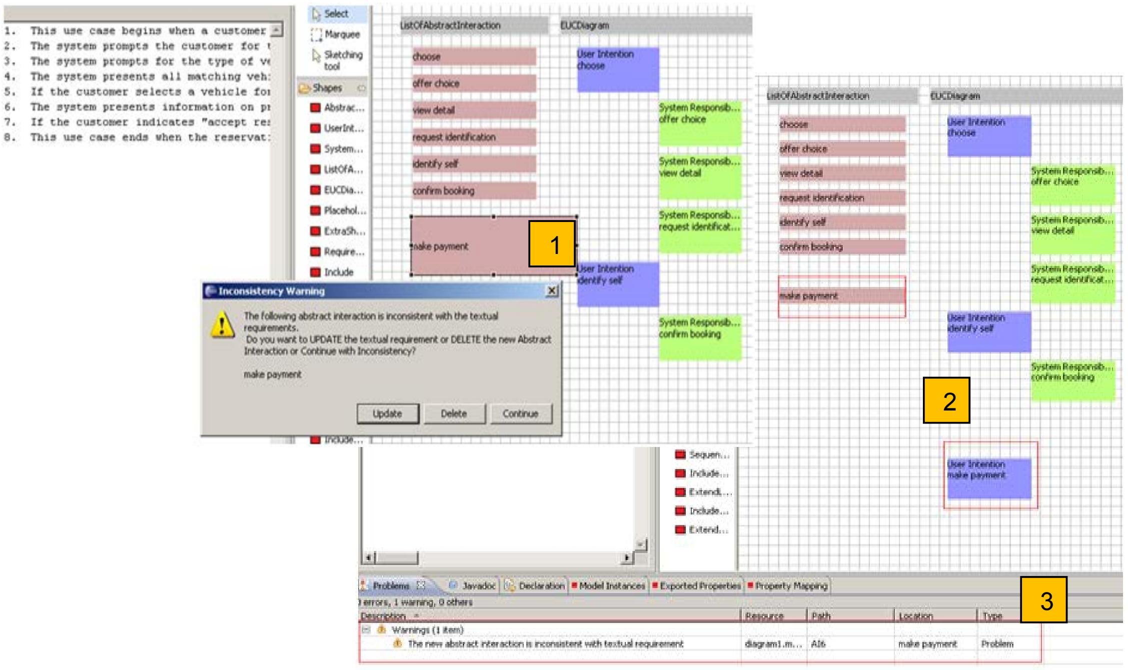
Massila, a requirement engineer, would like to validate the requirements that she has collected from the client, John, who is the car rental information manager. To do this, as shown in Figure 8.2, she types in the requirements in a form of user scenario to the textual editor or copies them in from an existing file (1) and has the tool trace the essential requirements (abstract interactions) (2). Here, she verifies the list of abstract interactions provided by the tool and then has the tool generate the EUC model (3). In order to check for the consistency and dependencies among the EUC component and the abstract interaction and the user scenario, she performs trace back by using the event handler from the EUC component or abstract interaction. For trace back (as shown in Figure 8.2), the selected EUC component (A) and its associated abstract interaction (B) changes colour to red and the associated essential interactions (C) are highlighted with “***”. The processes of tracing forward/backward and mapping are assisted by event handlers. These tracings show and maintain the consistency among the requirement components.



```
1.  This use case begins when a customer indicates he wis
2.  The system ***prompts the customer for the pickup***
3.  The system ***prompts for the type***       ehicle the
4.  The system presents all matching vehi       available a
5.  If the customer selects a vehicle for rental, the sys
6.  The system presents information on protection product
7.  If the customer indicates "accept reservation," the s
8.  This use case ends when the reservation confirmation
```

Figure 8. 2: Capturing requirements - trace the abstract interaction, trace back and map to EUC model

By using MaramaAI, Massila can make any modification to any of the requirement components if she is not satisfied with the results provided by the tool. For example, if she thinks one of the abstract interactions is missing, she could add a new abstract interaction to the list. In particular, she might think that an abstract interaction “make payment” is missing from the list. Thus, she adds a new abstract interaction “make payment” to the list. This action triggers an inconsistency warning and the options either to update, delete or continue without updating the textual natural language requirements to appear to inform her that an inconsistency has occurred in the requirement components (as shown in Figure 8.3: (1)). She then chooses to continue without updating the user scenario as she probably thinks that the “make payment” abstract interaction is necessary and matches the user scenario. Although the option “continue” is chosen by her, she can still map the newly-added abstract interaction to the EUC model (2). This triggers a problem marker to inform her of the inconsistency error for later consideration to resolve the inconsistency (3).



```
1.  This use case begins when a customer
2.  The system prompts the customer for t
3.  The system prompts for the type of ve
4.  The system presents all matching veh:
5.  If the customer selects a vehicle fo:
6.  The system presents information on p:
7.  If the customer indicates "accept re:
8.  This use case ends when the reservat:
```

Inconsistency Warning

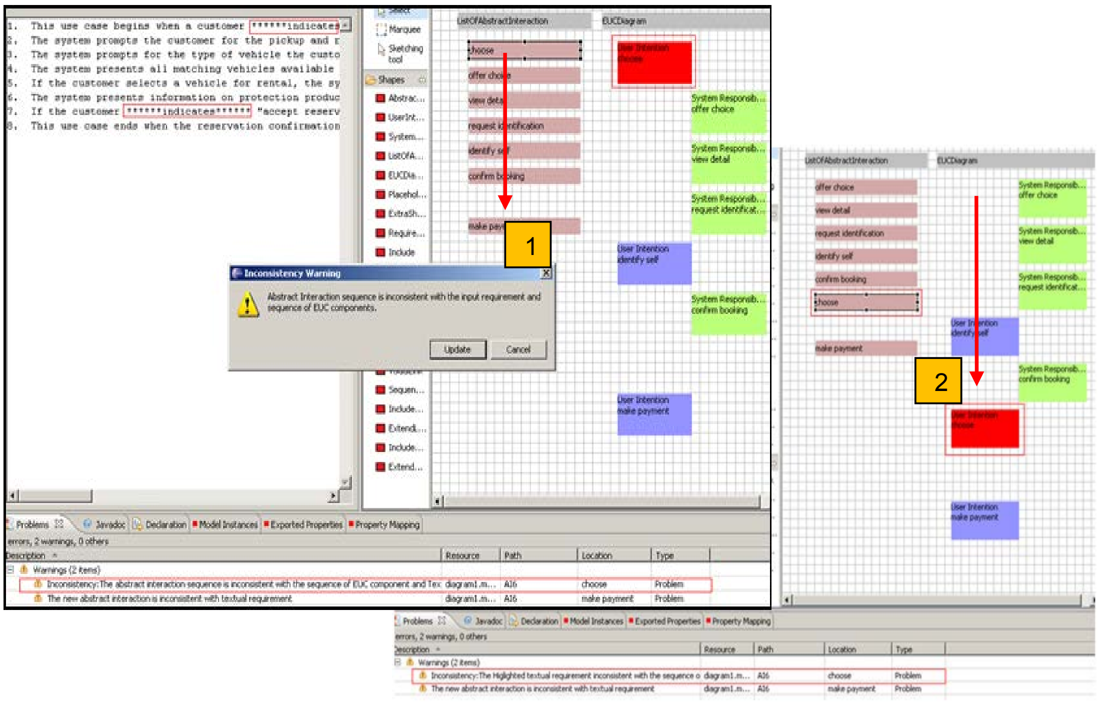
The following abstract interaction is inconsistent with the textual requirements.
Do you want to UPDATE the textual requirement or DELETE the new Abstract Interaction or Continue with Inconsistency?

make payment

```
0 errors, 1 warning, 0 others
Description                                                      Resource      Path   Location       Type
⊟ Warnings (1 item)
     The new abstract interaction is inconsistent with textual requirement   diagram1.m...  AI6    make payment   Problem
```

Figure 8. 3: Add New Item to Abstract Interaction

Next, Massila is also unhappy with the sequence ordering of one the abstract interaction components: "choose". She thinks this abstract interaction should be above the "make payment" component as shown in Figure 8.4 (1) because the user should choose from the option before any payment should be requested. This triggers the associated EUC component "choose" to change colour to red and the essential interaction "indicates" to be highlighted with"***". An Inconsistency warning also appears to inform her of the inconsistencies and provide options either to update or cancel the change. A problem marker also provides warning on inconsistencies that still exist. Then she decides to update the sequence ordering, and this automatically also changes the position of the EUC component" choose" (2). However, the ordering of the highlighted essential interactions is not altered as such changes could affect the structure of the user scenario. This action also triggers a problem marker to warn about the inconsistencies that have not been completely resolved.


ListOfAbstractInteraction
EUCDiagram
choose
offer choice
view detail
request identification
identify self
confirm booking
make payment
User Intention choose
System Responsib... offer choice
System Responsib... view detail
System Responsib... request identificat...
User Intention identify self
System Responsib... confirm booking
User Intention make payment
1
2
Marquee
Sketching tool
Shapes
Update
Cancel

```
1.  This use case begins when a customer ******indicates
2.  The system prompts the customer for the pickup and r
3.  The system prompts for the type of vehicle the custo
4.  The system presents all matching vehicles available
5.  If the customer selects a vehicle for rental, the sy
6.  The system presents information on protection produc
7.  If the customer ******indicates****** "accept reserv
8.  This use case ends when the reservation confirmation
```

Inconsistency Warning

Abstract Interaction sequence is inconsistent with the input requirement and sequence of EUC components.

Problems | Javadoc | Declaration | Model Instances | Exported Properties | Property Mapping

errors, 2 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (2 items)				
Inconsistency:The abstract interaction sequence is inconsistent with the sequence of EUC component and Tex	diagram1.m...	AI6	choose	Problem
The new abstract interaction is inconsistent with textual requirement	diagram1.m...	AI6	make payment	Problem

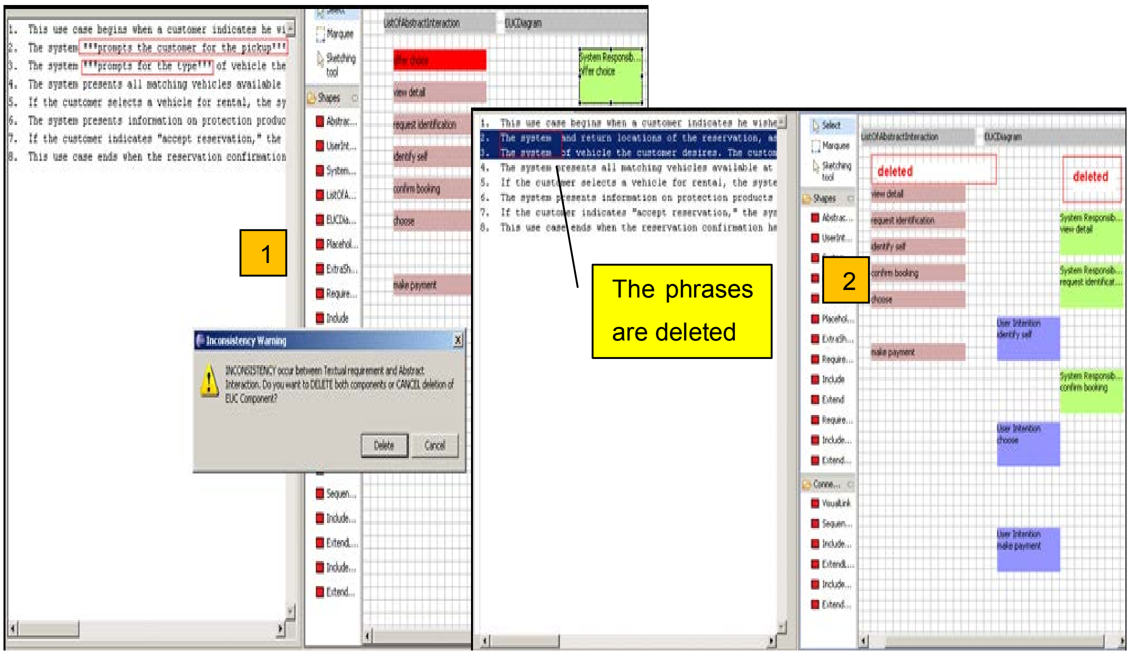
Problems | Javadoc | Declaration | Model Instances | Exported Properties | Property Mapping

errors, 2 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (2 items)				
Inconsistency:The Higlighted textual requirement inconsistent with the sequence o	diagram1.m...	AI6	choose	Problem
The new abstract interaction is inconsistent with textual requirement	diagram1.m...	AI6	make payment	Problem

Figure 8. 4: Change of Abstract Interaction Sequence Ordering

On reviewing the extracted EUC, Massila feels that there is an extra component in the EUC model. She thinks that the EUC component “offer choice” is not necessary and needs to be deleted. She believes there is a redundancy between the “choose” and “offer choice” component. Thus, she selects the “offer choice” component to be deleted. This action triggers the associated abstract interaction to automatically change colour to red and the associated essential interactions “prompts the customer for the pickup” and “prompts for the type” to be highlighted with “***”. The inconsistency warning also appears to inform the inconsistencies and options to either delete or cancel the deletion. Although a notification of the inconsistencies is provided, she still thinks she needs to delete the “offer choice” component. This triggers the associated abstract interaction and essential interactions also to be deleted. This occurs as the tool tries to keep all the three requirement components in a consistent state.


1. This use case begins when a customer indicates he wi
2. The system ***prompts the customer for the pickup***
3. The system ***prompts for the type*** of vehicle the
4. The system presents all matching vehicles available
5. If the customer selects a vehicle for rental, the sy
6. The system presents information on protection produc
7. If the customer indicates "accept reservation," the
8. This use case ends when the reservation confirmation

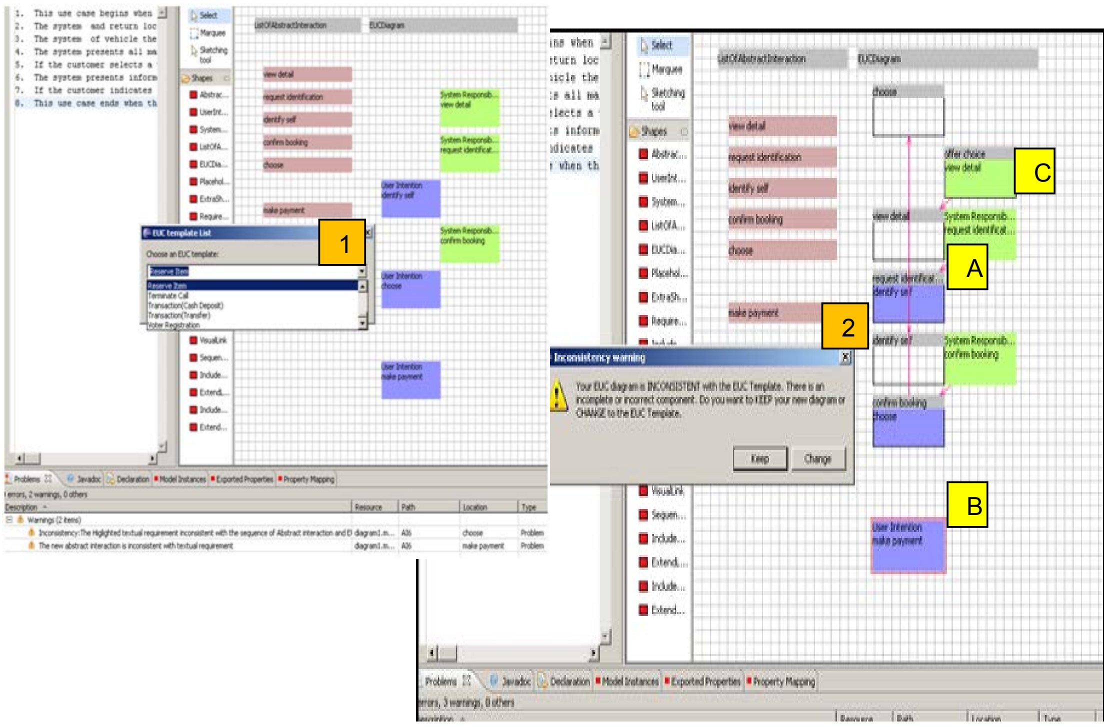
Inconsistency Warning

INCONSISTENCY occur between Textual requirement and Abstract Interaction. Do you want to DELETE both components or CANCEL deletion of EUC Component?

1. This use case begins when a customer indicates he wishe
2. The system and return locations of the reservation, a
3. The system of vehicle the customer desires. The custo
4. The system presents all matching vehicles available at
5. If the customer selects a vehicle for rental, the syste
6. The system presents information on protection products
7. If the customer indicates "accept reservation," the sys
8. This use case ends when the reservation confirmation ha

Figure 8. 5: Delete the EUC component

Being a novice requirement engineer, Massila is keen to validate her extracted EUC model against a best-practice EUC template. Thus, she looks through the list of available templates and chooses the pattern “Reserve Item” as shown in Figure 8.6 (1) that appears to be similar to this scenario. She matches the pattern to her EUC model and sees that she has missed some interactions as a few sequence orderings and components are incorrect. In addition, an extra component also exists in the interaction. As shown in Figure 8.6 (2), the incorrect sequence ordering is shown by the red visual links (A), the existence of the extra component “make payment” (B) is outlined with red and the correct component “offer choice” (C) is shown by a grey element on top of the green shape “view detail” which also displays the incorrect component and position held by the “view detail” component. As there is an unmatched interaction between the generated EUC and the best- practice template, Massila is notified with an inconsistency warning and given options to either keep or change the generated EUC following the best-practice template. She agrees with the warning and the errors shown. She then selects to change this EUC model to the EUC interaction templates.



Inconsistency warning

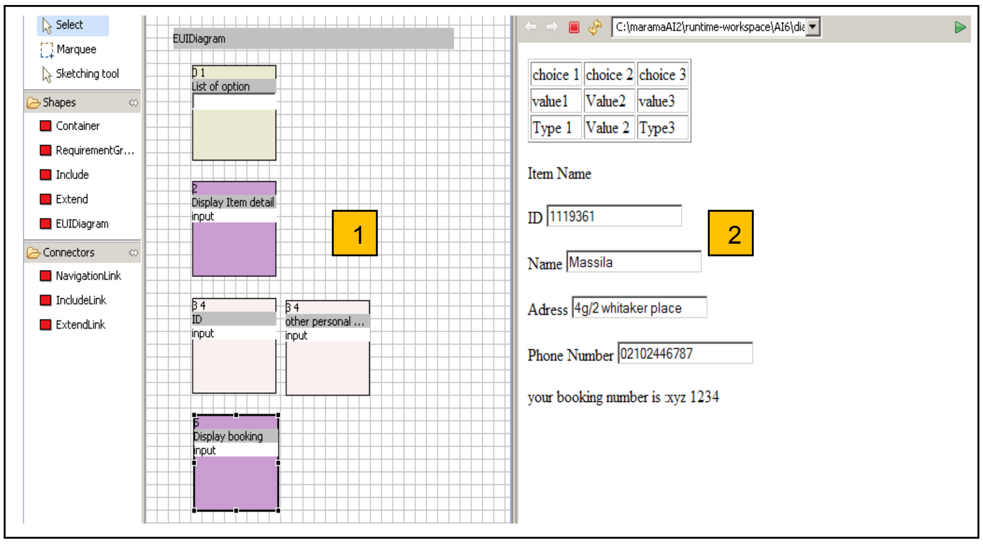
Your EUC diagram is INCONSISTENT with the EUC Template. There is an incomplete or incorrect component. Do you want to KEEP your new diagram or CHANGE to the EUC Template.

errors, 2 warnings, 0 others

Description	Resource	Path	Location	Type
Warnings (2 items)				
Inconsistency:The Highlighted textual requirement inconsistent with the sequence of Abstract interaction and E	diagram1.m...	A06	choose	Problem
The new abstract interaction is inconsistent with textual requirement	diagram1.m...	A06	make payment	Problem

Figure 8. 6: Visual differencing to check for incorrectness and incompleteness

When Massila is satisfied with the requirements components, she sits with John to validate the requirements and to confirm the consistency of her captured requirements with the earlier requirements provided by John. In order to allow John to better understand the requirement components, she then has the tool map the EUC model to abstract prototype: EUI prototype as (1) and also has the tool translate EUI prototype to a concrete UI view in a HTML form (2) as shown in Figure 8.7.



choice 1	choice 2	choice 3
value1	Value2	value3
Type 1	Value 2	Type3

Item Name

ID 1119361

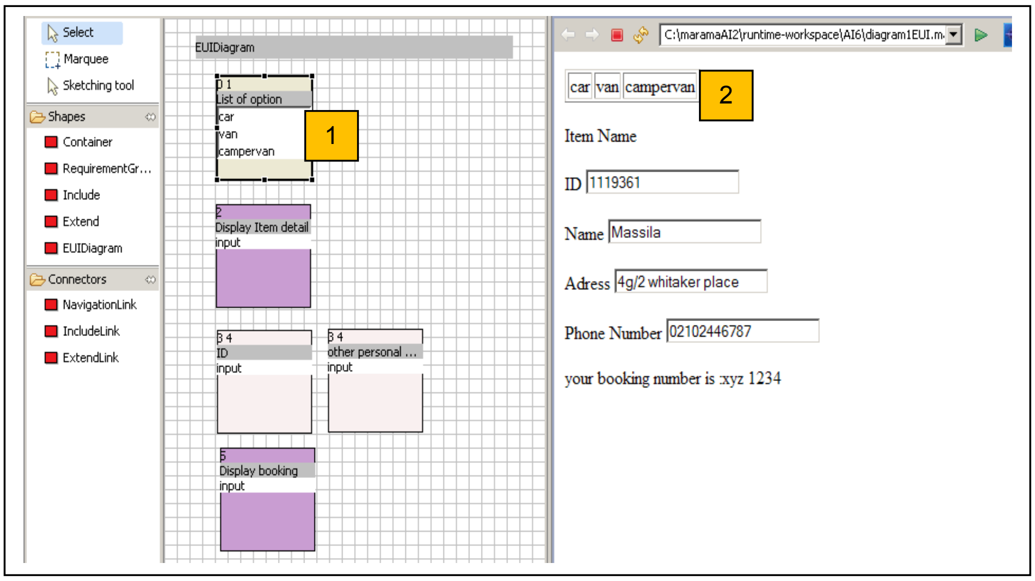
Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

your booking number is :xyz 1234

Figure 8. 7: The generated EUI prototype (1) and translated HTML form (2)



car	van	campervan

Item Name

ID 1119361

Name Massila

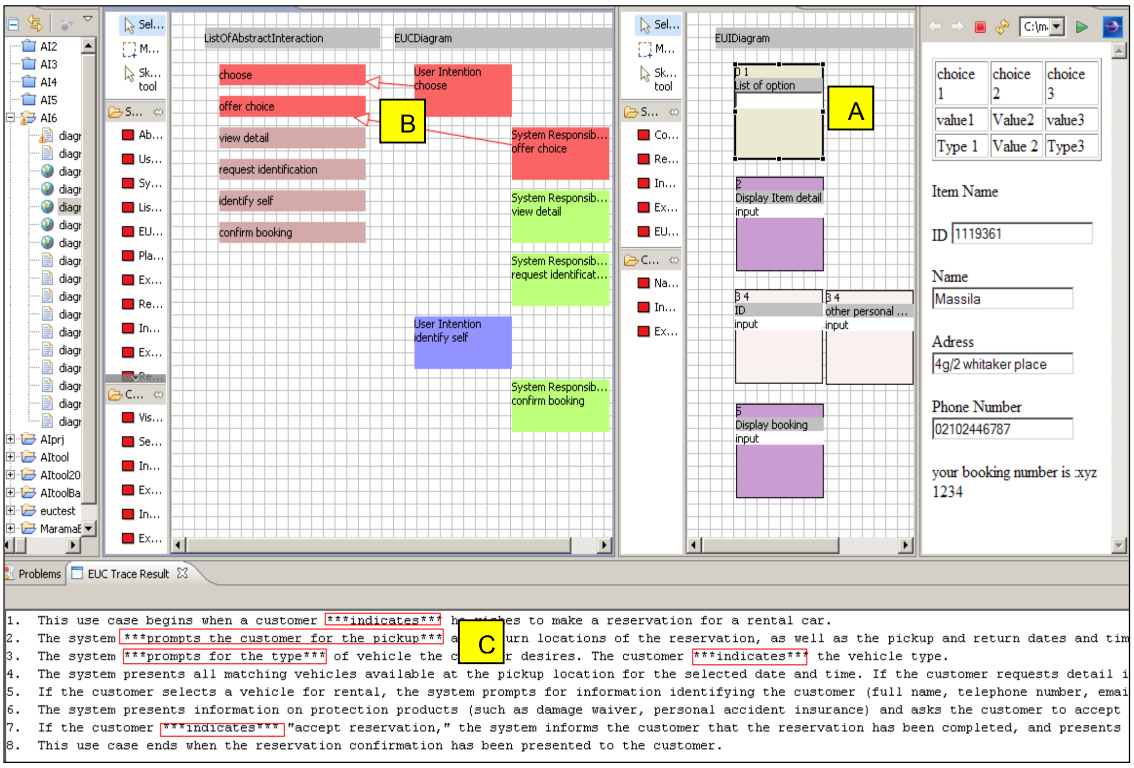
Adress 4g/2 whitaker place

Phone Number 02102446787

your booking number is :xyz 1234

Figure 8. 8: Modifications in Prototypes

From the walkthrough, John thinks that the EUI component of "List of options" is a bit vague and would be better understood by adding detail of the types of options such as "car, van and campervan" as shown in Figure 8.8(1). Massila modifies that on the spot and then shows the result in a HTML form as in Figure 8.8(2). Next, she wants to validate and confirm the consistency of her point of view against John's point of view. She selects one of the EUI components "List of options" (A) and has the tool trace back to the other requirement components: EUC model, abstract interactions and textual natural language requirements as shown in Figure 8.9. This triggers the associated EUC component and abstract interactions “choose and offer choice” (B) to change colour to red and the essential interactions “indicates, prompts the customer for the pickup and prompts for the type” (C) of the user scenario to be highlighted. Here, Massila is able to confirm the consistency of all requirement components with John for the earlier collected requirements.


ListOfAbstractInteraction
EUCDiagram
EUIDiagram
choose
offer choice
view detail
request identification
identify self
confirm booking
User Intention choose
System Responsib... offer choice
System Responsib... view detail
System Responsib... request identificat...
User Intention identify self
System Responsib... confirm booking
List of option
Display Item detail input
ID input
other personal ... input
Display booking input
A
B
C
Problems
EUC Trace Result

choice 1	choice 2	choice 3
value1	Value2	value3
Type 1	Value 2	Type3

Item Name

ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

your booking number is :xyz 1234

```
1.   This use case begins when a customer ***indicates*** h    es to make a reservation for a rental car.
2.   The system ***prompts the customer for the pickup*** a    urn locations of the reservation, as well as the pickup and return dates and tim
3.   The system ***prompts for the type*** of vehicle the c    r desires. The customer ***indicates*** the vehicle type.
4.   The system presents all matching vehicles available at the pickup location for the selected date and time. If the customer requests detail i
5.   If the customer selects a vehicle for rental, the system prompts for information identifying the customer (full name, telephone number, emai
6.   The system presents information on protection products (such as damage waiver, personal accident insurance) and asks the customer to accept
7.   If the customer ***indicates*** "accept reservation," the system informs the customer that the reservation has been completed, and presents
8.   This use case ends when the reservation confirmation has been presented to the customer.
```

Figure 8. 9: Trace back which performs from the EUI prototype

In summary, Massila has used the MaramaAI tool to capture automatically the abstract interactions and to extract the EUCs from the user scenario provided by John. She also used the tool to manage the consistency and to validate the incorrectness and incompleteness of the requirements by using the essential interaction pattern library and “best- practice” template from the EUC interaction pattern library, together with the inconsistency warning, problem marker and highlights. She then sat with John to verify and confirmed further the consistency of the requirements by having the tool generate the prototypes: EUI prototype and HTML form.

8.3 Case Study 2: Book Check-out in a Library System

We choose the Library Book Borrowing (LLB) system as the second example of a requirement to demonstrate the key features of our tool in handling the use of extension/include in the use case. This user scenario is written by Sendall and Strohmeier [2] as a set of use cases which is also as a case study for the Software Engineering Education project (SWEED).

This automated LLB system is developed to ease the task of librarians in processing book loans in all the departmental libraries of any university [2]. The system makes use of the available library book search system to undertake the book search.

Users do not need to identify themselves to the system to search for a book, but this is required when they want to check-out a book, to check their loan status or to reserve a book which is “on hold”. This process of identification is conducted by using a card and a password together with password verification for security reasons, similar to the ATM system.

All books are provided with barcodes. A barcode scanner is used to check out the book. If any failures happen to the scanner, the barcodes need to be manually entered. The description of the user scenario we have chosen is shown in Figure 8.10.

LBB System
Use Case: check-out books
Scope: Library Book Borrowing System
Level: User Goal
Intention in Context: The intention of the User is to check-out books from a library. Only one User can check-out books at any one time.
Primary Actor: User (becomes Member once s/he has identified him/herself with the System)

Main Success Scenario:
1. User requests System to check-out books.
2. User identifies him/herself to System.
Step 3 is repeated for each book that is checked-out by Member.
3. Member registers book with System.
4. Member indicates to System that s/he has finished checking out books.
5. System records all books registered by Member as on loan, requests Printer to print out a receipt* for the session, and puts itself in a state to receive the next User.

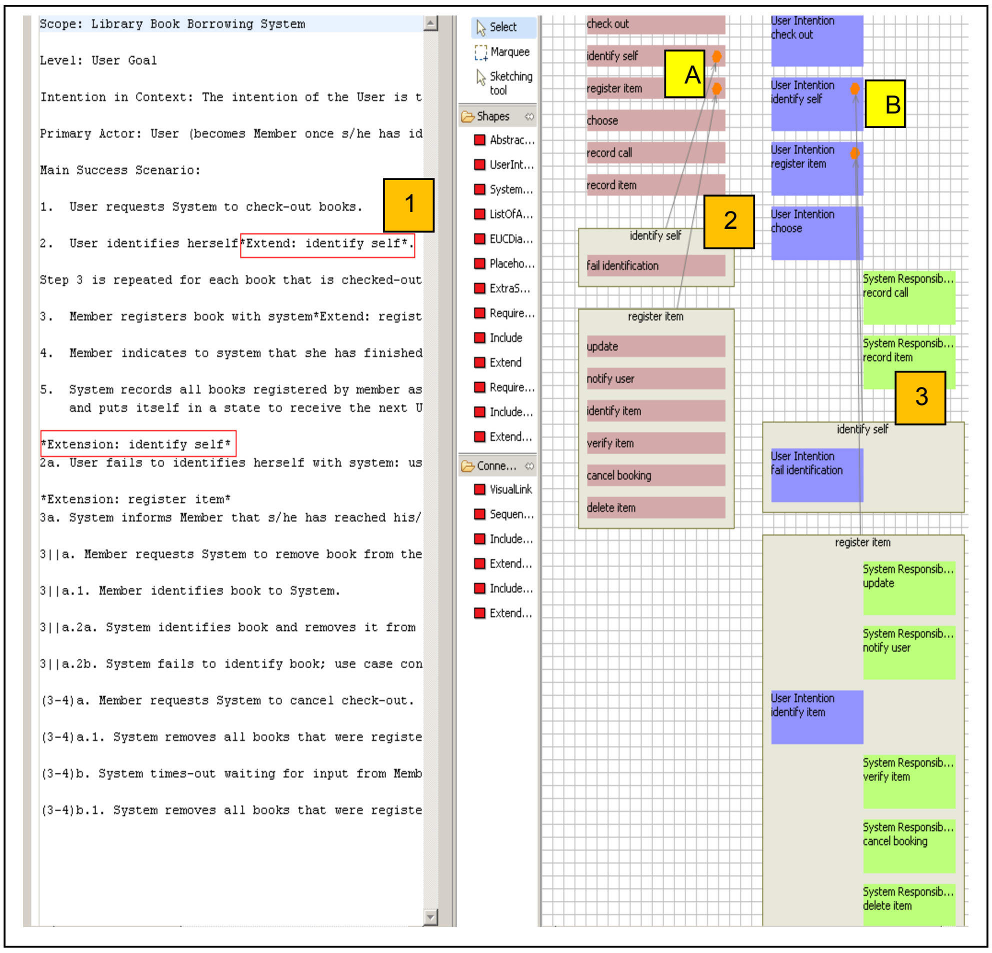
Extensions:
2a. User fails to identify him/herself with System: use case ends in failure.
3a. System informs Member that s/he has reached his/her maximum number of books allowed on loan; use case continues at step 4.
3||a. Member requests System to remove book from the books that are checked out:
3||a.1. Member identifies book to System.
3||a.2a. System identifies book and removes it from the list of books registered by Member; use case continues from where it was interrupted.
3||a.2b. System fails to identify book; use case continues from where it was interrupted.
(3-4)a. Member requests System to cancel check-out.
(3-4)a.1. System removes all books that were registered by Member, and puts itself in a state to receive the next User; use case ends in failure.
(3-4)b. System times-out waiting for input from Member:
(3-4)b.1. System removes all books that were registered by Member, and puts itself in a state to receive the next User; use case ends in failure.

Figure 8. 10: Example of User Scenario in a Form of Use Case Description: Check-out books of a LLB system [2]

8.3.1 Example of Usage

We demonstrate our tool key features which deal with extension/include in the use case description for the requirements as below:

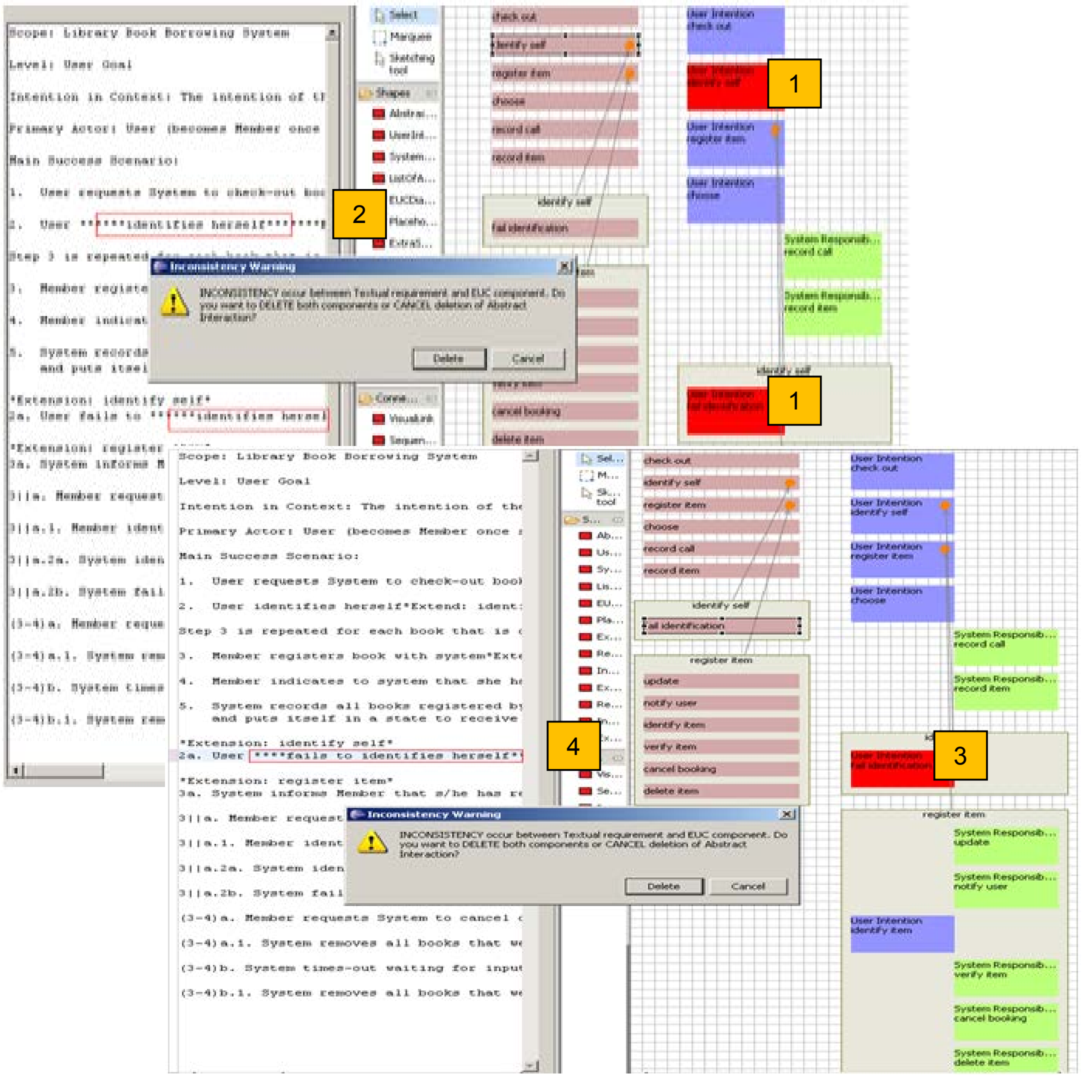
Massila, the requirement engineer, meets Johan, who is the Library IT Support Manager, and gathers the requirements for the library booking system. After she collects the requirements, she refines the user scenario and describes it in a form of use case description. She also thinks that the use case description for the functional requirements needs to be supported with the use of extension. She then types in the requirements written in a form of use case description to the textual editor, following the guidelines provided by the tool as shown in figure 8.11 (1). Then she has the tool trace the abstract interaction and map to the EUC model. The process of tracing the abstract interaction and EUC component and the process of mapping the abstract interaction to the EUC model are conducted similarly to the first example. The tool generates a separate component for the extended abstract interactions (2). A small orange circle with a grey extension link (A) is also generated to show the extension. The generated abstract interactions are then mapped to the EUC component with an extension (3). A similar separate component and a small orange circle with grey extension links (B) are also provided for the EUC model to show the extension.



```
Scope: Library Book Borrowing System

Level: User Goal

Intention in Context: The intention of the User is t

Primary Actor: User (becomes Member once s/he has id

Main Success Scenario:

1.  User requests System to check-out books.

2.  User identifies herself*Extend: identify self*.

Step 3 is repeated for each book that is checked-out

3.  Member registers book with system*Extend: regist

4.  Member indicates to system that she has finished

5.  System records all books registered by member as
    and puts itself in a state to receive the next U

*Extension: identify self*
2a. User fails to identifies herself with system: us

*Extension: register item*
3a. System informs Member that s/he has reached his/

3||a. Member requests System to remove book from the

3||a.1. Member identifies book to System.

3||a.2a. System identifies book and removes it from

3||a.2b. System fails to identify book; use case con

(3-4)a. Member requests System to cancel check-out.

(3-4)a.1. System removes all books that were registe

(3-4)b. System times-out waiting for input from Memb

(3-4)b.1. System removes all books that were registe
```

Figure 8. 11: Capturing requirements-trace the abstract interaction and map to EUC model

From the results shown by the tool, Massila thinks she needs to delete an abstract interaction “identify self” as she feels that this component is not important to the requirement. Thus, she selects the “identify self” component to be deleted. This action triggers the associated EUC components “identify self” and “ fail identification” to automatically change colour to red (1), and the associated essential interaction “identifies herself” to be highlighted with “***” as shown in Figure 8.12(2). The EUC component “fail identification” is also highlighted because this component is an extension to the main abstract interaction “identify self” component. The inconsistency warning also appears to inform the inconsistencies and options to either delete or cancel the deletion. From the notification on the inconsistencies, she thinks she has made a wrong decision. So she cancels the deletion. This triggers the associated EUC component and essential interactions to revert to the original format. Then she identifies another component which actually needs to be deleted: “fail identification” component is the one that needs to be deleted. Thus, she selects “fail identification” to be deleted. She recognises that different situations happen. Here, only the “fail identification” of an EUC component changes colour to red (3) and only the essential interaction “fails to identifies herself” which is association with “fail identification”, is highlighted with *** (4). This happens because the “fail identification” component is an extension component which does not affect the main abstract interaction if any deletion happens.


```
Scope: Library Book Borrowing System

Level: User Goal

Intention in Context: The intention of the

Primary Actor: User (becomes Member once

Main Success Scenario:

1.   User requests System to check-out book

2.   User identifies herself*Extend: ident:

Step 3 is repeated for each book that is

3.   Member registers book with system*Exte

4.   Member indicates to system that she he

5.   System records all books registered by
     and puts itself in a state to receive

*Extension: identify self*
2a. User ****fails to identifies herself*

*Extension: register item*
3a. System informs Member that s/he has re

3||a. Member request

3||a.1. Member ident

3||a.2a. System iden

3||a.2b. System fail

(3-4)a. Member requests System to cancel

(3-4)a.1. System removes all books that we

(3-4)b. System times-out waiting for input

(3-4)b.1. System removes all books that we
```

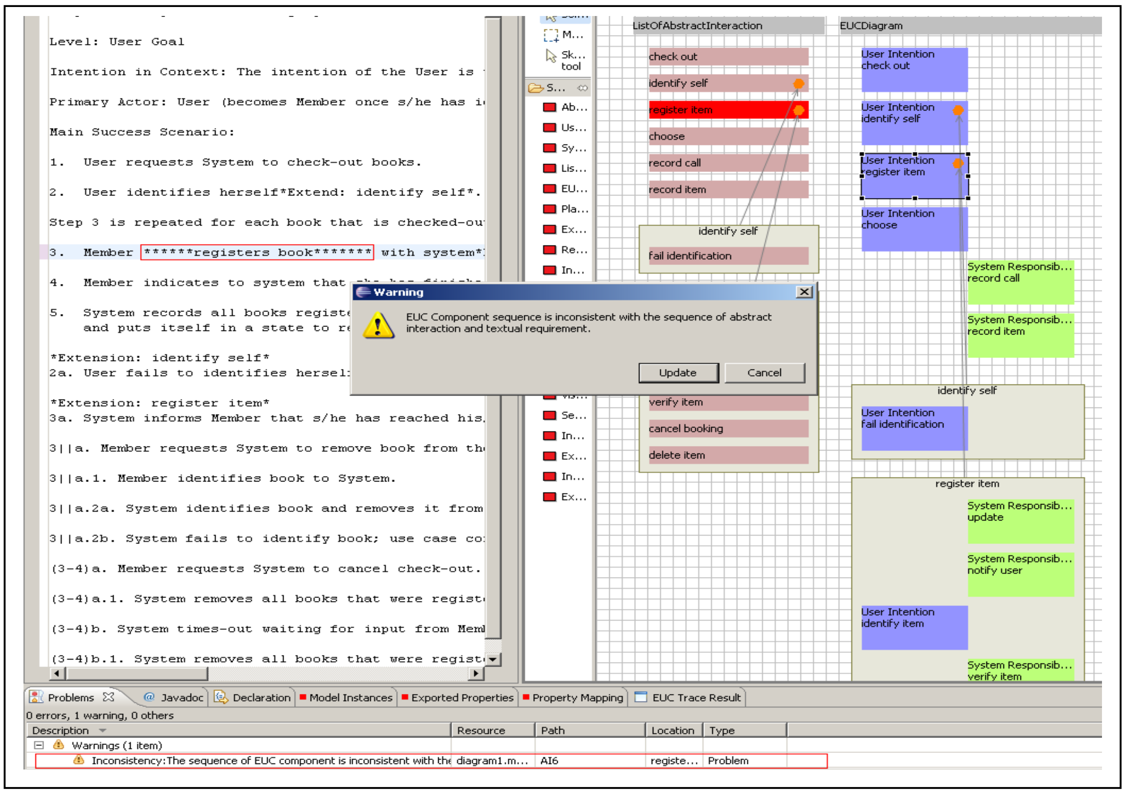
Inconsistency Warning

INCONSISTENCY occur between Textual requirement and EUC component. Do you want to DELETE both components or CANCEL deletion of Abstract Interaction?

Delete
Cancel
Select
Marquee
Sketching tool
Shapes
check out
identify self
register item
choose
record call
record item
fail identification
update
notify user
identify item
verify item
cancel booking
delete item
User Intention check out
User Intention identify self
User Intention register item
User Intention choose
System Responsib... record call
System Responsib... record item
User Intention fail identification
System Responsib... update
System Responsib... notify user
User Intention identify item
System Responsib... verify item
System Responsib... cancel booking
System Responsib... delete item
1
2
3
4

Figure 8. 12: Deletion of Abstract Interaction

On reviewing the requirements, Massila also thinks that the sequence ordering of one of the EUC component, “register item”, needs to be changed to another position above the “check out” component as shown in Figure 8.13. This triggers the associated abstract interaction component “register item” to change colour to red and the essential interaction “registers book” to be highlighted with “***”. An Inconsistency warning appears to inform Massila of the inconsistencies and provides options either to update or cancel the change. Further, a problem marker also appears to warn of inconsistencies that still exist. Despite these notifications, she still thinks that she needs to change the sequence ordering of the “register item” component. This automatically changes the position of the abstract interaction component “register item”. However, the ordering of the highlighted essential interactions is not changed. This is because she believes the change could affect the structure of the description. This also triggers the problem marker to warn of inconsistencies that have not been completely resolved. The ordering of the extension component “register item” is not changed. This is because the change of the sequence ordering only affects the interaction of the main EUC components and not the extension component.



```
Level: User Goal

Intention in Context: The intention of the User is

Primary Actor: User (becomes Member once s/he has i

Main Success Scenario:

1.   User requests System to check-out books.

2.   User identifies herself*Extend: identify self*.

Step 3 is repeated for each book that is checked-ou

3.   Member ******registers book******* with system*

4.   Member indicates to system that

5.   System records all books registe
     and puts itself in a state to re

*Extension: identify self*
2a. User fails to identifies hersel

*Extension: register item*
3a. System informs Member that s/he has reached his,

3||a. Member requests System to remove book from th

3||a.1. Member identifies book to System.

3||a.2a. System identifies book and removes it from

3||a.2b. System fails to identify book; use case co

(3-4)a. Member requests System to cancel check-out.

(3-4)a.1. System removes all books that were regist

(3-4)b. System times-out waiting for input from Mem

(3-4)b.1. System removes all books that were regist
```

Warning

EUC Component sequence is inconsistent with the sequence of abstract interaction and textual requirement.

Update Cancel

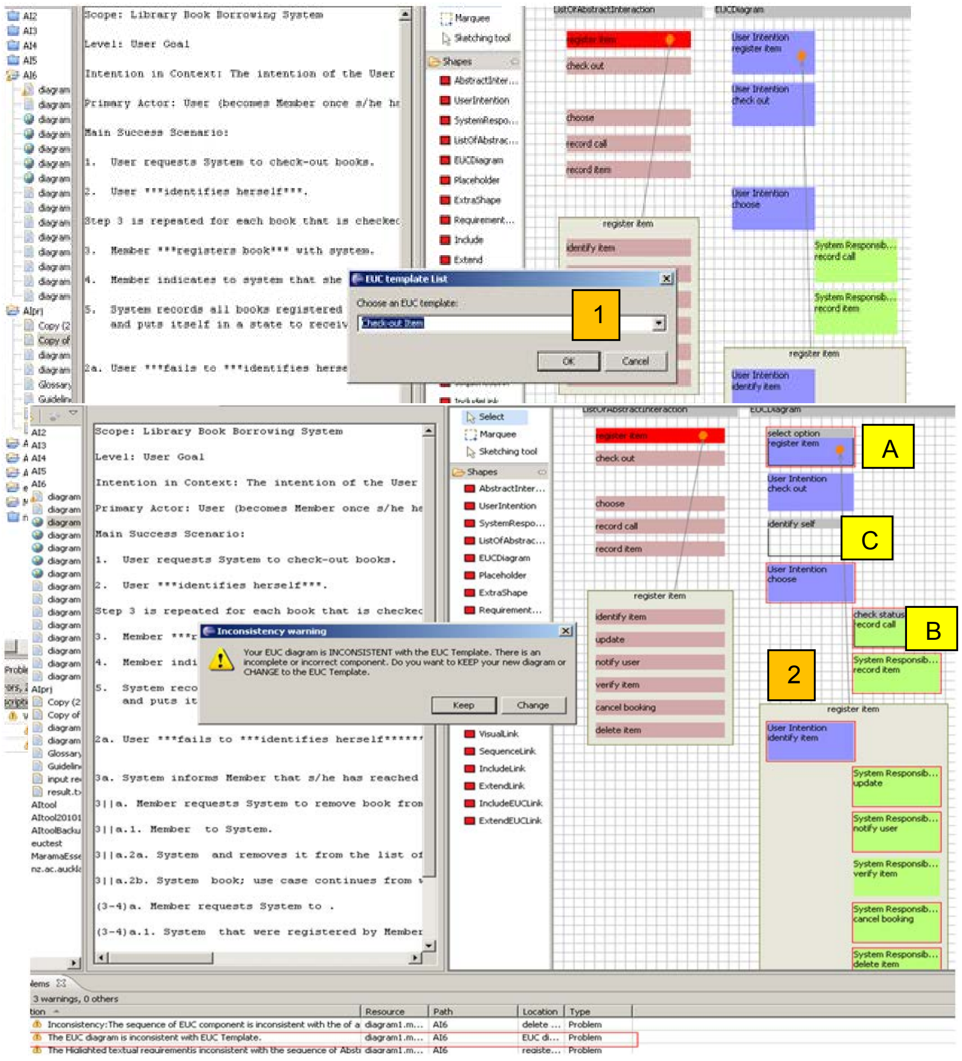
Problems | Javadoc | Declaration | Model Instances | Exported Properties | Property Mapping | EUC Trace Result

0 errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
Inconsistency:The sequence of EUC component is inconsistent with the	diagram1.m…	AI6	registe…	Problem

Figure 8. 13: Change of EUC component Sequence Ordering

After making these changes, Massila thinks she should further validate the interactions in the requirements against the best-practice template of EUC interactions. She selects a template which looks very similar to her requirement descriptions. From the available templates, she chooses the pattern “Checkout Item” as shown in Figure 8.14 (1). She matches the pattern to her EUC model and sees that she has missed some interactions as there are a few extra and incorrect components in her model. Figure 8.14 (2) shows the existence of extra components such as “register item, choose, record call, record item, identify item, update, notify user, verify item, cancel booking and delete” outlined in red, and the correct components, such as “select option”, are shown by a grey element on top of the blue shape of “register item” (A), while the “check status” is shown by a grey element on top of the green shape of “record call” (B). These two components show that there are missing and incorrect components “register item” and “check status”. There is also an incomplete interaction where an EUC component “identify self” (C) needs to be provided to complete the interaction. As there is an unmatched interaction between the generated EUC and the best-practice template, Massila is notified with an inconsistency warning and options to either keep or change the generated EUC as per the best-practice template. She then chooses to keep this EUC model which triggers a problem marker to notify her of the inconsistency that still occurs between the generated EUC model and the EUC interaction templates.



```
Scope: Library Book Borrowing System

Level: User Goal

Intention in Context: The intention of the User

Primary Actor: User (becomes Member once s/he he

Main Success Scenario:

1.   User requests System to check-out books.

2.   User ***identifies herself***.

Step 3 is repeated for each book that is checkec

3.   Member ***registers book*** with system.

4.   Member indicates to system that she

5.   System records all books registered
     and puts itself in a state to receiv

2a. User ***fails to ***identifies herself******

3a. System informs Member that s/he has reached

3||a. Member requests System to remove book from

3||a.1. Member  to System.

3||a.2a. System  and removes it from the list of

3||a.2b. System  book; use case continues from w

(3-4)a. Member requests System to .

(3-4)a.1. System  that were registered by Member
```

EUC template List

Choose an EUC template:

Check-out Item

Inconsistency warning

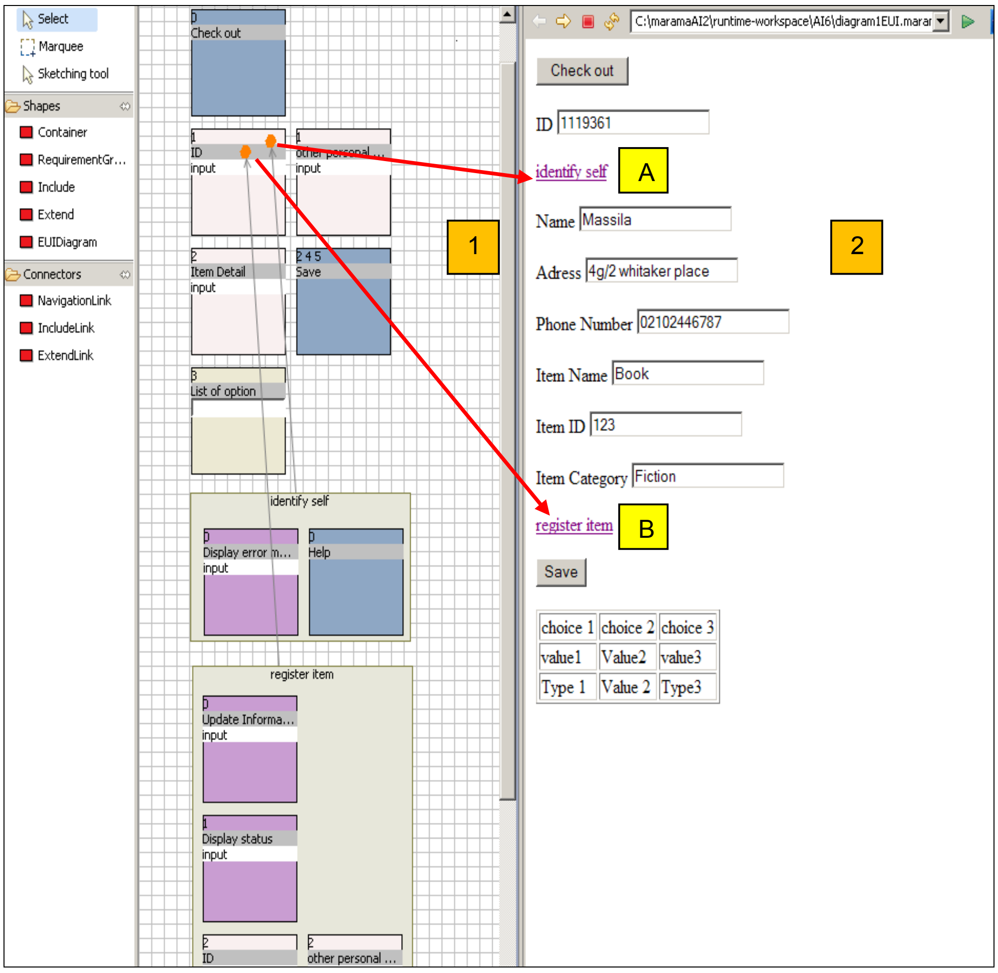
Your EUC diagram is INCONSISTENT with the EUC Template. There is an incomplete or incorrect component. Do you want to KEEP your new diagram or CHANGE to the EUC Template.

3 warnings, 0 others

```
tion                                                                       Resource      Path  Location   Type
Inconsistency:The sequence of EUC component is inconsistent with the of a  diagram1.m…   AI6   delete …   Problem
The EUC diagram is inconsistent with EUC Template.                         diagram1.m…   AI6   EUC di…    Problem
The Higlighted textual requirementis inconsistent with the sequence of Abst diagram1.m…  AI6   registe…   Problem
```

Figure 8. 14: Visual differencing to check for incorrectness and incompleteness

In order to further validate her captured requirements and to ease the discussion process with Johan, she has the tool map the validated EUC model to the low-fidelity prototype - EUI prototype in Marama EUI editor as shown in Figure 8.15 (1). Here, she sits with Johan and together they walk through the captured requirements and the UI. She finds that the generated EUI prototype in Marama EUI editor is also supported with extension components similar to the Marama Essential as shown in Figure 8.15 (1). She then has the tool translate the EUI prototype to a more concrete UI view in a HTML form (2) to give a better picture of the captured requirements to Johan. The tool generates the HTML page with hyperlinks (underlined purple words) (A and B) in an HTML form. These hyperlinks are from the extension components in the EUI prototype (1).



Check out

ID 1119361

identify self

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

Item Name Book

Item ID 123

Item Category Fiction

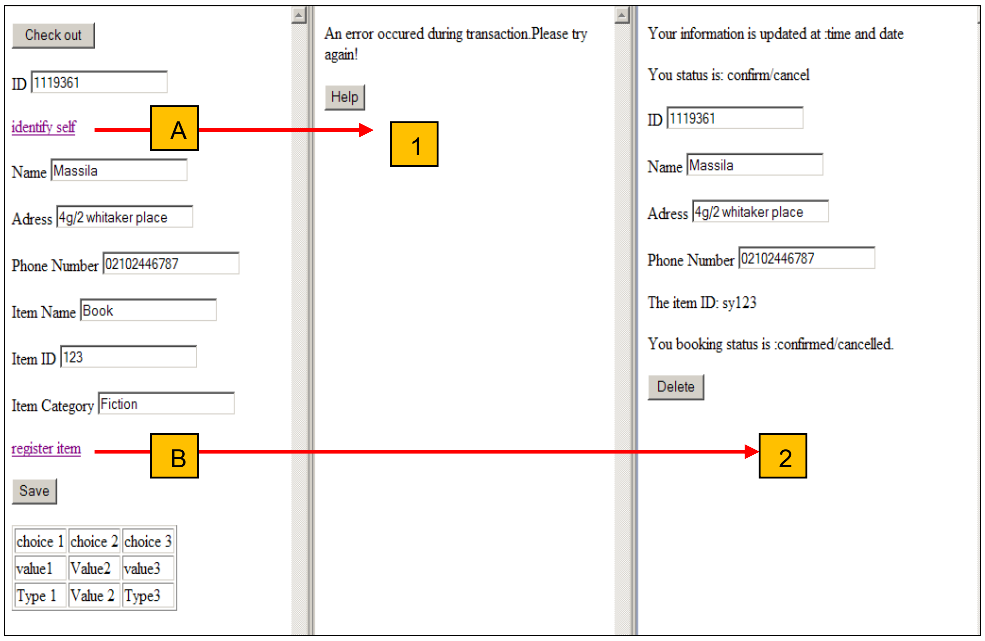
register item

Save

choice 1	choice 2	choice 3
value1	Value2	value3
Type 1	Value 2	Type3

Figure 8. 15: EUI Prototype with the extension components (1) and the generated HTML form with hyperlinks (2)

After viewing the HTML page, Johan would like to see the results of the hyperlink. Massila shows him how each hyperlink associates with the main page. Illustrated in Figure 8.16, the arrow (A) shows the navigation of hyperlink “identify self” to page 1 and the arrow (B) shows the navigation of hyperlink “register item” to page 2.


Check out

ID 1119361

identify self

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

Item Name Book

Item ID 123

Item Category Fiction

register item

Save

choice 1	choice 2	choice 3
value1	Value2	value3
Type 1	Value 2	Type3

An error occured during transaction.Please try again!

Help

Your information is updated at :time and date

You status is: confirm/cancel

ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

The item ID: sy123

You booking status is :confirmed/cancelled.

Delete

Figure 8. 16: HTML main page and hyperlink pages generated from EUI prototype

From these views, Massila tries to confirm the consistency of the captured requirements on the LLB system from her viewpoint as against Johan's original idea. Johan is also happy as he could visualise his requested requirements in a way he understood.

In summary, Massila has used the MaramaAI tool to capture automatically the abstract interactions and to extract the EUCs from the use case descriptions provided by Johan. The tool demonstrates its ability to support the use of extension/include in a use case. She also used the tool to manage the consistency and to validate the incorrectness and incompleteness of the requirements by using the essential interaction pattern library and "best-practice" template from the EUC interaction pattern library together with the inconsistency warning, problem marker and highlights. She then sat with John to verify and confirm further the consistency of the requirements by having the tool generate the prototypes with hyperlinks for both the EUI prototype and HTML form.

8.4 Case Study 3: Manage Events with Event Listing System

We chose the Silicon Dreams Event Listing System (www.reaction.co.nz), a real-world example with several requirements, as the third example of a requirement to demonstrate the key features of our tool in handling multiple requirements. This user scenario is obtained from a real industry project to manage events from Silicon Dream Ltd, a web design company which specialises in developing websites, e-commerce and online marketing [155]. It has developed an event-listing website since 1999 but closed that site due to legal, technical and environmental issues which badly affected the progress and usage of the website. It is now trying to re-launch this website with new technology and a more interactive portal. The company also wants to make sure it is correctly developed and fits the right end-users.

The Silicon Dreams Event Listing System is an event-management website for adults. It allows users to browse available event lists by event category such as night-life, eating out and stage. Users can also select and view events and venues. This website also allows users to register, to receive newsletters and to post comments and feedback. This portal is also for the administration to update, edit and maintain the database on the venues and events as well as to manage the list of events, reports and reviews. All the requirements for this website were collected from Mark Young, the project manager, who has kindly allowed us to use them in this study. He described them in the form of a user scenario which we then presented in the form of use case description. For this example, we will only describe two different parts of the requirements: Manage Venue and Manage Event Review. The requirement descriptions are shown in Figure 8.17.

Scope: Event Listing System (Manage Venue)

Summary: user (super admin, city admin, venue admin) is the only authorised person to manage venues. For event listing purposes, event venues management is necessary.

1. User logs into the system {www.recation.co.nz} with authorised admin policy.
2. User clicks on “venue button” which opens an interface with “Add Venue”, ”Modify Venue”, and “Delete Venue” options.
3. User performs add venue. User enters the venue name.
4. System verifies the venue.
5. User enters the venue information.
6. System stores the venue into database.

Extension:

4.1a If venue does not exists: the system displays an error message.

Scope: Event Listing System (Manage Event Review)

Summary: user (super admin, city admin) would like to review an event posted by registered viewer to avoid violent reviews. In addition, they also need to block (de-activate) the user (portal viewer) account to avoid future inconvenience from that user (portal viewers).

1. User logs in the system {www.recation.co.nz} with authorised admin policy.
2. User chooses “reviews on event” option, which opens an interface with posted reviews of specific event.
3. User reviews the comment on a particular event.

Extension:

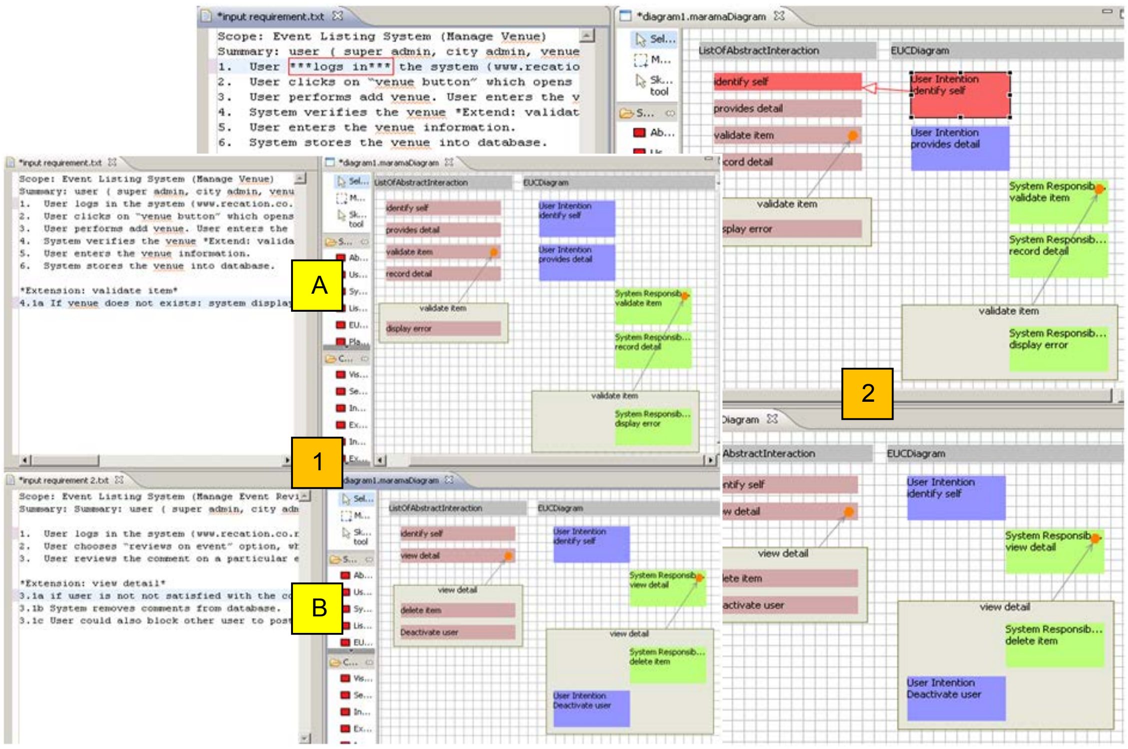
3.1a If user is not satisfied with the comment or the comment seems to be violent, he could remove the comments.
3.1b System removes comments from database.
3.1c User could also block other user’s ability to post comments by clicking the “De-Activate user” button.

Figure 8. 17: Example of User Scenarios in the form of Use Case Descriptions: Manage Venue and Manage Event review from the Silicon Dreams Event Listing System specification

8.4.1 Example of Usage

We demonstrate our tool key features with this user scenario which needs to deal with multiple requirements.

Massila, the requirement engineer, meets Mark, the project manager, and gathers the requirements for the event-listing system. After she collects the requirements, she refines the user scenarios and describes them in a form of use case descriptions. She then types in the requirements written in the form of use case descriptions using the textual editor, following the guidelines provided by the tool as shown in figure 8.18 (1). Here, she tries to validate two scenarios at a time, editing each in a separate editor window. Then she has the tool trace the abstract interactions and map the abstract interactions to the EUC model. The process is conducted along the same lines as the first and second examples. The only difference is that she can view the results (A and B) for both scenarios and have the tool trace the abstract interaction sand map to the EUC models at the same time. She then has the tool trace back the EUC component "identify self". For this, as shown in Figure 8.18 (2), she finds that the tool is able to perform trace back only for one set of requirements at once and not both simultaneously. However, she is happy as she can automatically capture the essential requirements and the interaction.


*input requirement.txt

```
Scope: Event Listing System (Manage Venue)
Summary: user ( super admin, city admin, venue
1.   User ***logs in*** the system (www.recatio
2.   User clicks on "venue button" which opens
3.   User performs add venue. User enters the v
4.   System verifies the venue *Extend: validat
5.   User enters the venue information.
6.   System stores the venue into database.
```

*input requirement.txt

```
Scope: Event Listing System (Manage Venue)
Summary: user ( super admin, city admin, venu
1.   User logs in the system (www.recation.co.
2.   User clicks on "venue button" which opens
3.   User performs add venue. User enters the
4.   System verifies the venue *Extend: valida
5.   User enters the venue information.
6.   System stores the venue into database.

*Extension: validate item*
4.1a If venue does not exists: system display
```

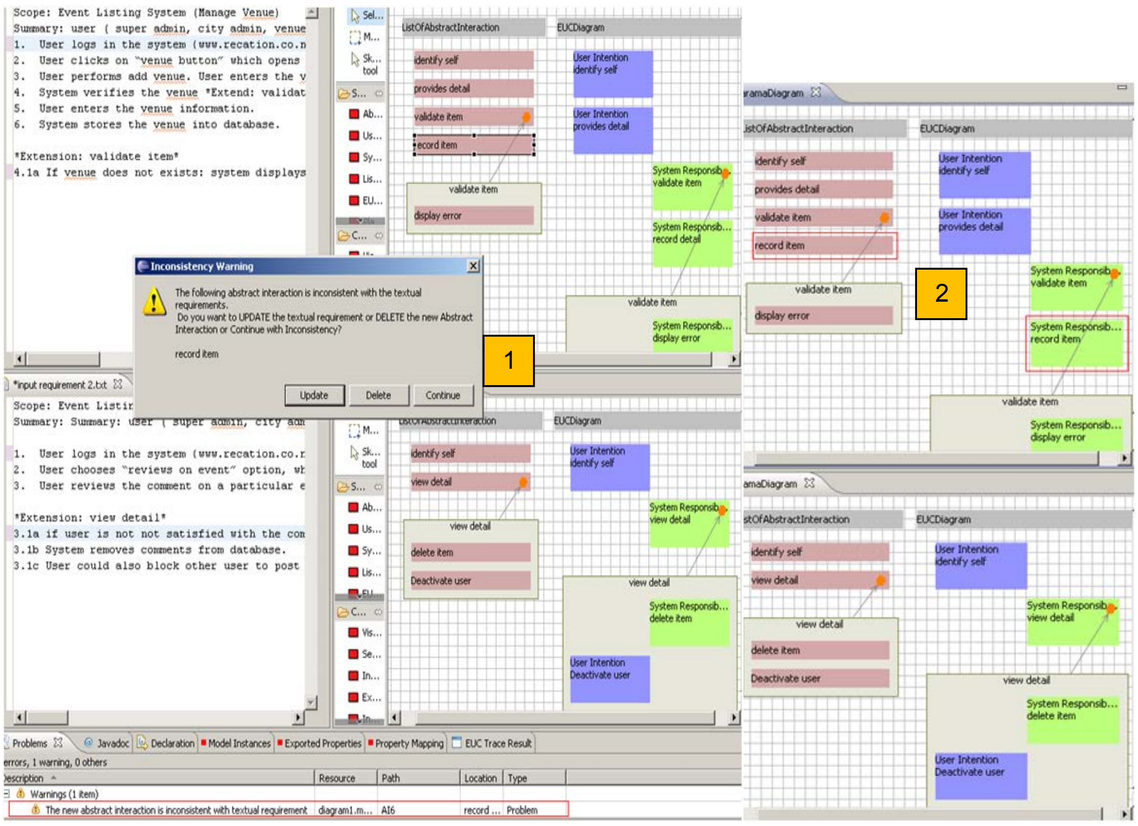
*input requirement 2.txt

```
Scope: Event Listing System (Manage Event Revi
Summary: Summary: user ( super admin, city adn

1.   User logs in the system (www.recation.co.r
2.   User chooses "reviews on event" option, wh
3.   User reviews the comment on a particular e

*Extension: view detail*
3.1a if user is not not satisfied with the co
3.1b System removes comments from database.
3.1c User could also block other user to post
```

*diagram1.maramaDiagram
ListOfAbstractInteraction
EUCDiagram
identify self
provides detail
validate item
record detail
validate item
display error
User Intention identify self
User Intention provides detail
System Responsib... validate item
System Responsib... record detail
validate item
System Responsib... display error
view detail
delete item
Deactivate user
System Responsib... view detail
view detail
System Responsib... delete item
User Intention Deactivate user
A
B
1
2

Figure 8. 18: Capturing Requirements - Trace the Abstract Interaction, Trace Back and Map to EUC model with Multiple Requirements

While reviewing the captured requirements model, Massila disagrees with the name of abstract interaction "record detail "provided by the tool for the first scenario: Manage event. Thus, she changes the name to "record item" and this automatically triggers an inconsistency warning to inform her of the inconsistency and provides options to update, delete or continue without updating the descriptions as shown in Figure 8.19 (1). Although, she is notified of the inconsistencies, she still thinks she has to continue with changing the name of the abstract interaction "record detail" to "record item”. So she selects “continue” from the options provided and this triggers a problem marker to provide the warning regarding the inconsistencies that still exist. She then maps the new abstract interaction to the EUC model. This also changes the name of the EUC component “record detail” to “record item” as shown in Figure 8.19 (2).



```
Scope: Event Listing System (Manage Venue)
Summary: user ( super admin, city admin, venue
1.  User logs in the system (www.recation.co.n
2.  User clicks on "venue button" which opens
3.  User performs add venue. User enters the v
4.  System verifies the venue *Extend: validat
5.  User enters the venue information.
6.  System stores the venue into database.

*Extension: validate item*
4.1a If venue does not exists: system displays
```

Inconsistency Warning

The following abstract interaction is inconsistent with the textual requirements.
Do you want to UPDATE the textual requirement or DELETE the new Abstract Interaction or Continue with Inconsistency?

record item

Update Delete Continue

*input requirement 2.txt

```
Scope: Event Listir
Summary: Summary: user ( super admin, city adm

1.  User logs in the system (www.recation.co.r
2.  User chooses "reviews on event" option, wh
3.  User reviews the comment on a particular e

*Extension: view detail*
3.1a if user is not not satisfied with the con
3.1b System removes comments from database.
3.1c User could also block other user to post
```

Problems | Javadoc | Declaration | Model Instances | Exported Properties | Property Mapping | EUC Trace Result

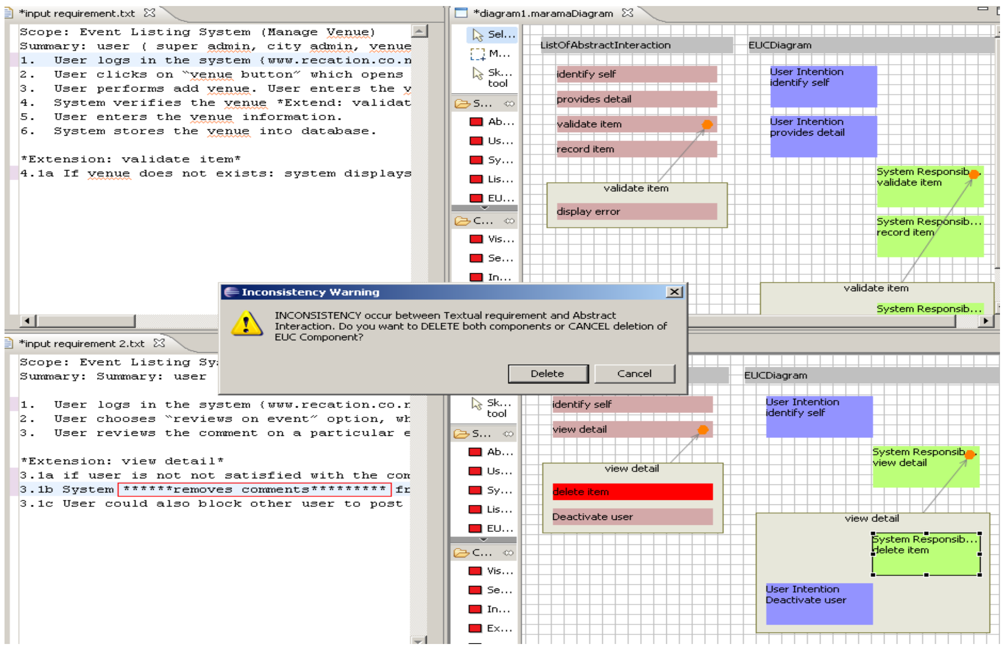
errors, 1 warning, 0 others

Description	Resource	Path	Location	Type
Warnings (1 item)				
The new abstract interaction is inconsistent with textual requirement	diagram1.m...	AI6	record ...	Problem

1
2
ListOfAbstractInteraction
EUCDiagram
identify self
provides detail
validate item
record item
display error
view detail
delete item
Deactivate user
User Intention identify self
User Intention provides detail
User Intention Deactivate user
System Responsib... validate item
System Responsib... record detail
System Responsib... record item
System Responsib... display error
System Responsib... view detail
System Responsib... delete item

Figure 8. 19: Change of Abstract Interaction Name

Massila is also unhappy with the “delete item” of the EUC component in the extension component of the second scenario. She considers deleting the “delete item” component. She selects the component and deletes it. This action triggers the associated abstract interaction to automatically change its colour to red and the associated essential interactions “to be highlighted with “***” as shown in Figure 8.20. An inconsistency warning also appears to inform the inconsistencies and options to either delete or cancel the deletion. After seeing the notifications, she thinks she has made a wrong decision. She selects to cancel the deletion, and the highlighted abstract interaction returns to the original state.



```
*input requirement.txt
Scope: Event Listing System (Manage Venue)
Summary: user ( super admin, city admin, venue
1.   User logs in the system (www.recation.co.n
2.   User clicks on "venue button" which opens
3.   User performs add venue. User enters the v
4.   System verifies the venue *Extend: validat
5.   User enters the venue information.
6.   System stores the venue into database.

*Extension: validate item*
4.1a If venue does not exists: system displays
```

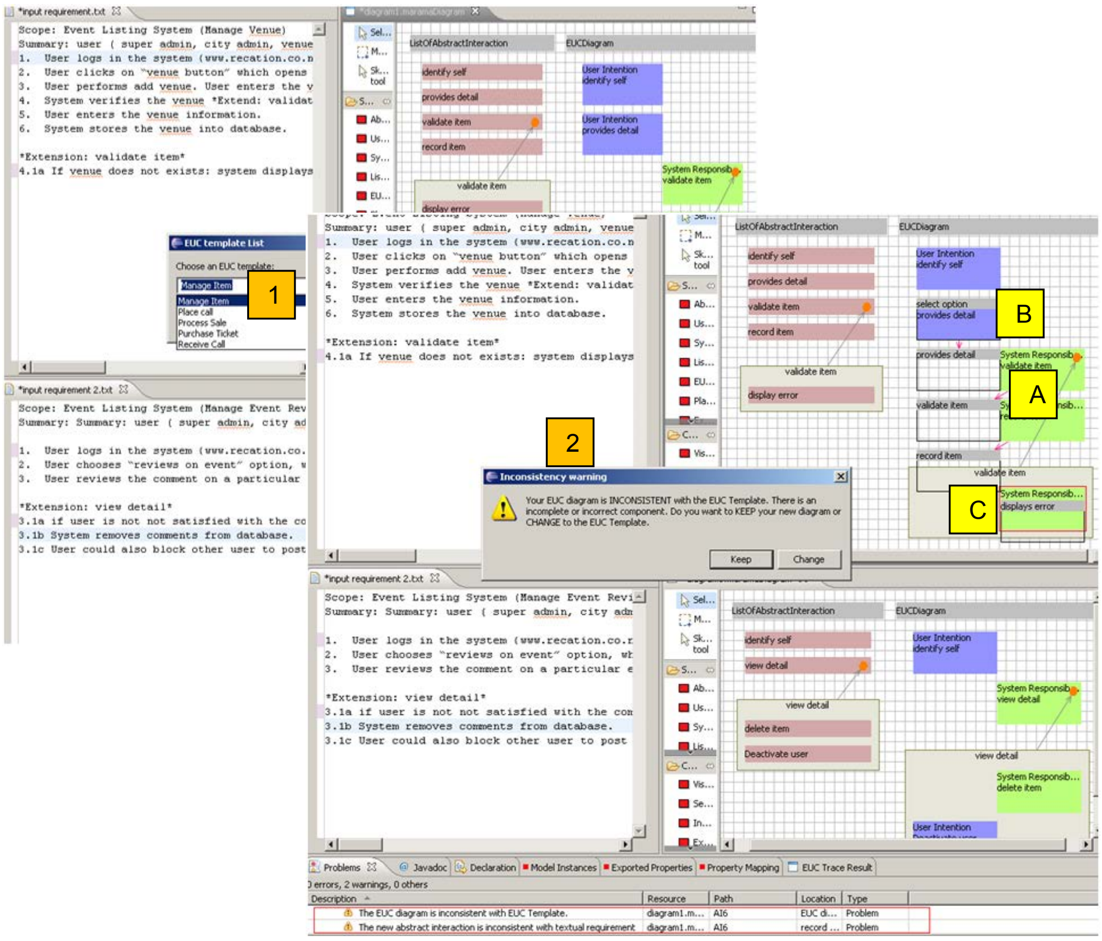
```
*input requirement 2.txt
Scope: Event Listing Sy
Summary: Summary: user

1.   User logs in the system (www.recation.co.r
2.   User chooses "reviews on event" option, wh
3.   User reviews the comment on a particular e

*Extension: view detail*
3.1a if user is not not satisfied with the com
3.1b System ******removes comments********* fr
3.1c User could also block other user to post
```

Inconsistency Warning

INCONSISTENCY occur between Textual requirement and Abstract Interaction. Do you want to DELETE both components or CANCEL deletion of EUC Component?

Figure 8. 20: Delete EUC component

Being a novice requirement engineer dealing with a project from an established company, Massila thinks she needs to further validate her requirements against a best-practice template, in order to ensure her requirements are correct and complete. Thus, she looks through the pattern catalogue and chooses pattern "Manage item" that appears to be very similar to the first scenario as shown in Figure 8.21 (1). She matches the pattern to her EUC model and sees that she has missed interactions as there are several incorrect sequence orderings and missing components. As shown in Figure 8.21 (2), an inconsistency warning to inform her of the inconsistency, incompleteness and incorrectness appears with options either to keep the designed model or change the model to follow the template. Here, the incorrect sequence ordering is displayed by the red visual links (A) showing the position held by each component. The missing component "select option" is identified by the grey element on top of the blue shape of "provides detail" (B). The component "display error" (C) is outlined in red and highlighted with grey as this shows that the template does not apply the extension component but agrees with the need of the "display error" component. Massila agrees with the template as she thinks that the tool has failed to trace the "select option" component. She thinks this may be because of the constraints faced by the essential interaction pattern library. However, she still thinks that the other components from her EUC model are correct. So, she would like to keep her EUC model as the tool does not support partial selection of the change and she subsequently has a further validation meeting with the client.


*input requirement.txt

```
Scope: Event Listing System (Manage Venue)
Summary: user ( super admin, city admin, venue
1.   User logs in the system (www.recation.co.n
2.   User clicks on "venue button" which opens
3.   User performs add venue. User enters the v
4.   System verifies the venue *Extend: validat
5.   User enters the venue information.
6.   System stores the venue into database.

*Extension: validate item*
4.1a If venue does not exists: system displays
```

*input requirement 2.txt

```
Scope: Event Listing System (Manage Event Revi
Summary: Summary: user ( super admin, city adm

1.   User logs in the system (www.recation.co.r
2.   User chooses "reviews on event" option, wh
3.   User reviews the comment on a particular e

*Extension: view detail*
3.1a if user is not not satisfied with the con
3.1b System removes comments from database.
3.1c User could also block other user to post
```

EUC template List

Choose an EUC template: Manage Item

Inconsistency warning

Your EUC diagram is INCONSISTENT with the EUC Template. There is an incomplete or incorrect component. Do you want to KEEP your new diagram or CHANGE to the EUC Template.

Keep | Change

Problems | Javadoc | Declaration | Model Instances | Exported Properties | Property Mapping | EUC Trace Result

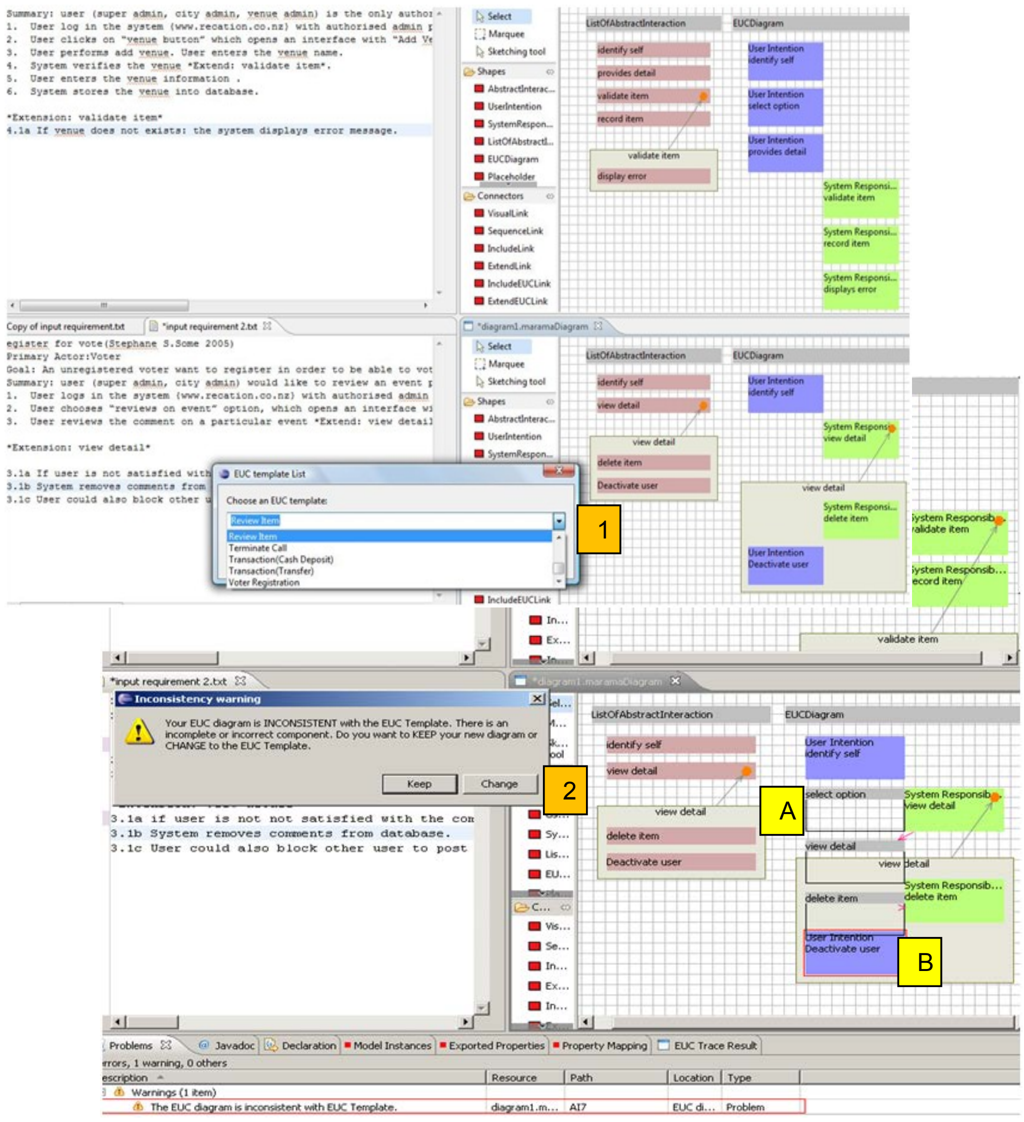
0 errors, 2 warnings, 0 others

Description	Resource	Path	Location	Type
The EUC diagram is inconsistent with EUC Template.	diagram1.m...	AI6	EUC di...	Problem
The new abstract interaction is inconsistent with textual requirement	diagram1.m...	AI6	record ...	Problem

Manage Item
Place call
Process Sale
Purchase Ticket
Receive Call
1
2
A
B
C
ListOfAbstractInteraction
EUCDiagram
identify self
provides detail
validate item
record item
display error
view detail
delete item
Deactivate user
User Intention identify self
User Intention provides detail
select option provides detail
System Responsib... validate item
System Responsib... displays error
System Responsib... view detail
System Responsib... delete item
User Intention

Figure 8. 21: Visual differencing to check for incorrectness and incompleteness for first scenario

Massila then thinks she should also further validate the second scenario. Thus, she looks through the list of available templates and chooses a pattern "Review Item" that seems to be very similar to the second scenario as shown in Figure 8.22 (1). She matches the pattern to her EUC model and sees that she has missed interactions as there are incorrect ordering, missing components and extra components in her EUC model. These are shown in Figure 8.22 (2): an inconsistency warning to warn her about the inconsistency, incompleteness and incorrectness appears with options either to keep the designed model or change the model following the template. Here, again, the incorrect sequence ordering is shown by the red visual link showing the position held by each component, the missing component "select option" (A) is identified by the grey element at the position after "identify self" and the extra component "Deactivate user" (B) is outlined in red. Massila disagrees with the template as she is confident with her EUC model. Thus, she then selects to keep her designed model.



```
Summary: user (super admin, city admin, venue admin) is the only author
1.  User log in the system (www.recation.co.nz) with authorised admin p
2.  User clicks on "venue button" which opens an interface with "Add Ve
3.  User performs add venue. User enters the venue name.
4.  System verifies the venue *Extend: validate item*.
5.  User enters the venue information .
6.  System stores the venue into database.

*Extension: validate item*
4.1a If venue does not exists: the system displays error message.
```

```
egister for vote(Stephane S.Some 2005)
Primary Actor:Voter
Goal: An unregistered voter want to register in order to be able to vot
Summary: user (super admin, city admin) would like to review an event p
1.  User logs in the system (www.recation.co.nz) with authorised admin
2.  User chooses "reviews on event" option, which opens an interface wi
3.  User reviews the comment on a particular event *Extend: view detail

*Extension: view detail*

3.1a If user is not satisfied with
3.1b System removes comments from
3.1c User could also block other u
```

EUC template List

Choose an EUC template:

Review Item

- Review Item
- Terminate Call
- Transaction(Cash Deposit)
- Transaction(Transfer)
- Voter Registration

Inconsistency warning

Your EUC diagram is INCONSISTENT with the EUC Template. There is an incomplete or incorrect component. Do you want to KEEP your new diagram or CHANGE to the EUC Template.

Keep | Change

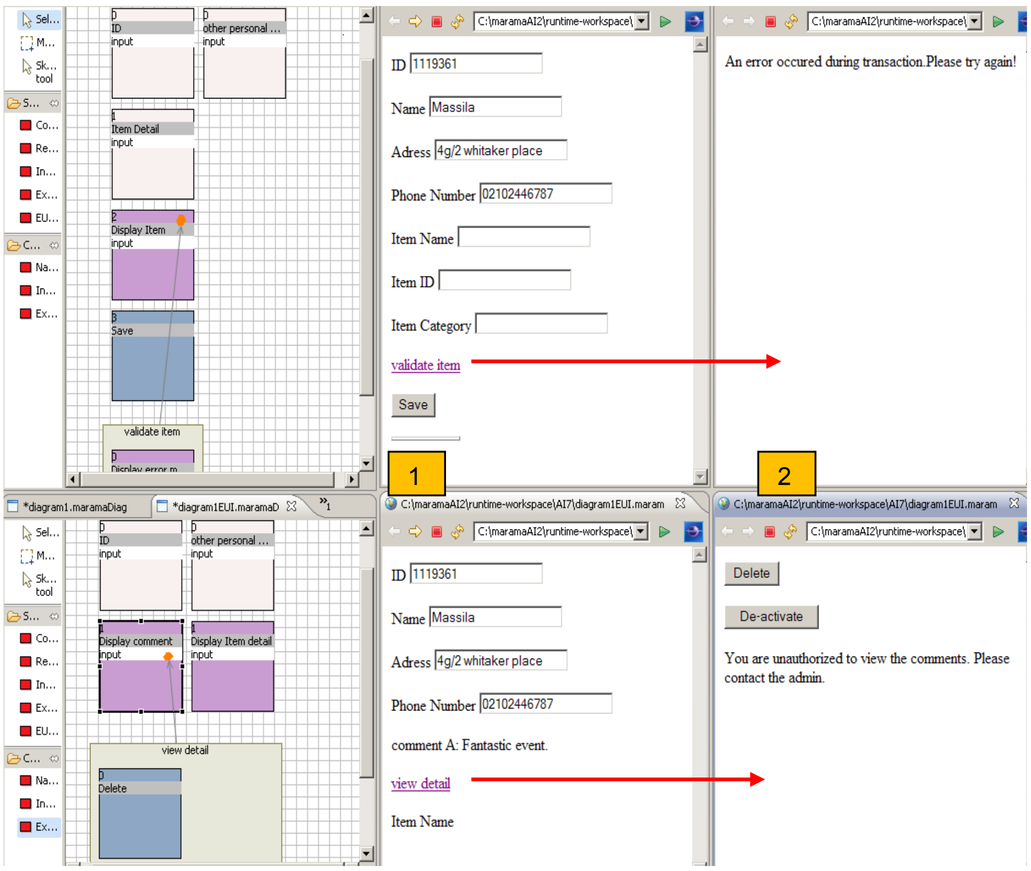
```
3.1a if user is not not satisfied with the com
3.1b System removes comments from database.
3.1c User could also block other user to post
```

errors, 1 warning, 0 others

escription	Resource	Path	Location	Type
Warnings (1 item)				
The EUC diagram is inconsistent with EUC Template.	diagram1.m...	AI7	EUC di...	Problem

Figure 8. 22: Visual differencing to check for incorrectness and incompleteness for second scenario

After Massila is satisfied with the requirements components, she meets Mark to validate further the requirements and to confirm the consistency of her captured requirements with the earlier requirements provided by Mark. In order to allow Mark to better understand the requirements components, she then has the tool map the EUC model to an abstract prototype: EUI prototype (1) and also has the tool translate the EUI prototype to a concrete UI view in a HTML form (2) as shown in Figure 8.23. They can view the prototypes for both scenarios. However, she and Mark subsequently walk through the requirements and the UI components for both sets of scenarios for better validation.



ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

Item Name

Item ID

Item Category

validate item

Save

An error occured during transaction.Please try again!

ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

comment A: Fantastic event.

view detail

Item Name

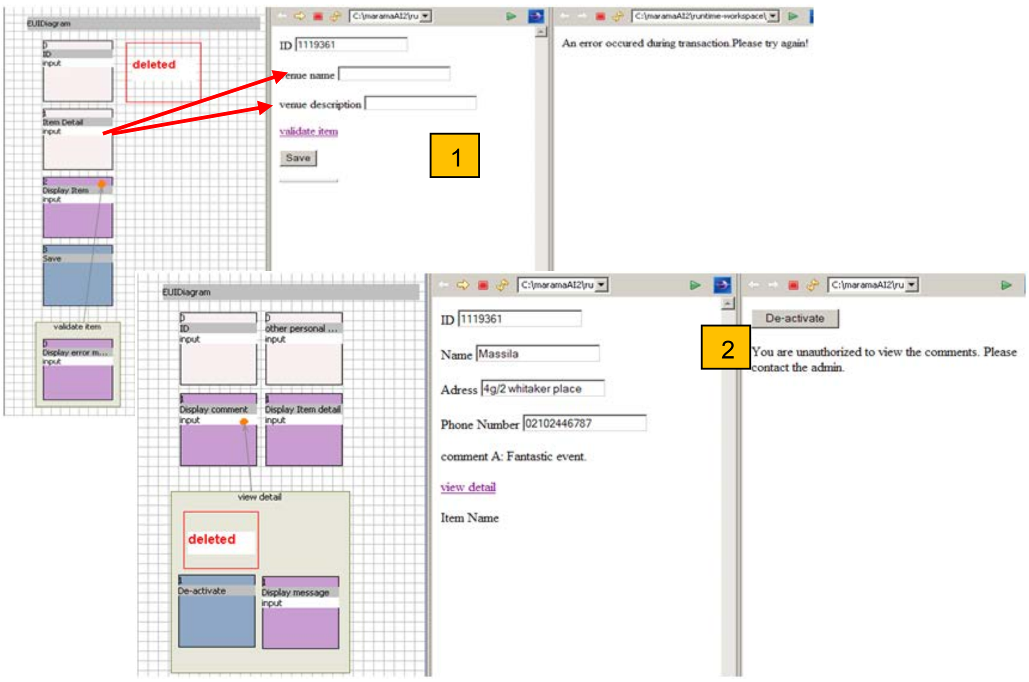
Delete

De-activate

You are unauthorized to view the comments. Please contact the admin.

Figure 8. 23: Multiple EUI prototypes and HTML forms

From the discussion, Mark thinks the EUI component “other personal detail” needs to be deleted and the EUI component “Item Detail” needs to be added with the information of “Venue Name and Venue Description”. Massila makes the changes requested by Mark and the results of the changes are shown in Figure 8.24(1). Then Mark also asks to delete the “delete” EUI component as he thinks that should not appear in the interface but as a process at the backend of the system. So Massila deletes the “delete” component and the result is shown in Figure 8.24 (2).



ID 1119361

venue name

venue description

validate item

Save

An error occured during transaction.Please try again!

ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

comment A: Fantastic event.

view detail

Item Name

De-activate

You are unauthorized to view the comments. Please contact the admin.

Figure 8. 24: Changes made to the EUI prototype (1) and the results in HTML form (2)

At the end of the process, Massila and Mark are satisfied with the help of MaramaAI as together they could confirm the consistency and validate the requirements. They are also happy as they could also finalise the requirements quickly and without delay.

In summary, Massila has used the MaramaAI tool to capture automatically the multiple abstract interactions and to extract multiple EUCs from the user scenario written in a form of use case descriptions provided by Mark. The tool demonstrates its ability to support multiple requirements together with the use of extension/include in a use case. She also used the tool to manage the consistency and validate the incorrectness and incompleteness of the multiple requirements by using the essential interaction pattern library and "best- practice" template from the EUC interaction pattern library together with the inconsistency warning, problem marker and highlights. She then sat with Mark to verify and confirm further the consistency of the multiple requirements by having the tool generate the prototypes: the EUI prototype and HTML form for both the requirements at once.

8.5 Discussion and Summary

We have applied our MaramaAI tool to three different domains of application: reserve a vehicle (rental car company), book check-out (Library Book Borrowing System) and event management (Silicon Dream Event Listing Website). We described our tool utilities by creating a user persona for each of the requirements examples which are described in a form of user scenario. We demonstrated each of the tool key features: capturing requirements, checking the consistency and validating the requirements and supporting the end-to-end rapid prototyping. We also described how each utility of the tool is interconnected.

The user scenario described in the first example is simple and straightforward. This scenario is used to demonstrate the basic utilities provided by our tool. The second user scenario is more complex as it is described with the use of extension. This scenario demonstrated that our tool is able to deal with more complex requirements and to deal with any type of requirement descriptions. The third user scenario was used to describe a real industry project requirement with several requirements. This demonstrated that our tool is able to be used in a real industry environment and showed how our tool simultaneously dealt with multiple requirements. Our main purpose in using these three different sets of requirements is to show that the utilities provided by MaramaAI can also be extended to a range of different domains and applications.

The demonstration of the tool also leads us to identify several limitations that we need to handle in future work. Firstly, we found that, the essential interaction library needs to be further enhanced as it does not trace an abstract interaction, which we believe is important, particularly in the third scenario. We believe though, that this can be solved as the process of updating the library is on-going. Next, we identified that the tool's utilities in validating the requirements' qualities using the visual differencing with "best-practice" templates need to be further enhanced. We noticed that, currently, the tool only allows the user to either convert or keep his/her original EUC model against the whole EUC interaction provided by the template and does not allow partial acceptance of a particular EUC interaction.

Further, our tool's utility in handling multiple requirements also need to be further enhanced. The tool can display all requirements together but the processes of consistency management and validation of requirements needs to be done one after the other. It would be preferable if both processes could be done simultaneously. However, this is a goal to be handled in our future work.

Chapter 9: Evaluation

This chapter presents the formal evaluation of our proof concept tool: MaramaAI (Automated Inconsistency Checker). The evaluation mechanism to evaluate the tool as well as the usability criteria and Cognitive Dimension notation (CD) used to evaluate the usability are also discussed. Then, discussion and comparison of the tool's evaluation results are also presented. This formal end user evaluation is approved by the University of Auckland Human Participants Ethics Committee (reference number: 2010/172).

9.1 Evaluation Mechanism Overview

We have conducted evaluations for three different phases of the prototype iteration with the same usability criteria and CD notation for our MaramaAI tool. This is a different evaluation from the informal evaluations presented in the earlier chapters, Chapter 4, 5 and 7. The earlier evaluations were used to inform refinements to the design, while the set of evaluations presented in this chapter took a larger group of participants (20) through each process step supported by the tool. The targeted end users for this evaluation are either postgraduate or undergraduate students who have sufficient background to understand software requirements. The participants of this survey were volunteers and their participation was treated anonymously. We recruited once, and the same group of participants was used to formally evaluate each phase, where phases corresponded to the development steps outlined in each of the previous three chapters. A full description of the evaluation phases is provided in appendices. The three separated evaluation phases are:

a. Part 1: Capturing requirements
   The participants were required to accomplish three parts of the evaluation. Firstly, to manually extract Essential Use Cases from the given scenario "reserve a vehicle" by Evan [1] and followed by repeating the same process automatically with MaramaAI. The participant was then given a set of questionnaires to be completed including open-ended feedback.
b. Part 2: Consistency Checking
   The participants were required to accomplish the following steps. Firstly, to explore the tool capabilities for checking the consistency of the requirements components: textual natural language requirement, abstract interactions and EUC model. The participant was asked to explore the tool by adding a new abstract interaction or textual requirement, delete any components or change the sequence of the components. Then, he/she was asked to explore the facility provided for validating other requirement qualities such as completeness and

correctness. Finally, participants were given a set of questionnaires to be completed including open-ended feedback.

c. Part 3: Exploring the end to end prototyping facility (End to End Rapid Prototyping)
   The participants were required to explore the refined tool capability for managing the requirements via an end-to-end prototyping facility. Here, participants were asked to map the three forms of requirements: textual natural language requirement, abstract interaction and EUC to the low fidelity UI in the form of Essential User Interface prototype (EUI prototype). Next, participants were asked to translate the EUI prototype to a concrete UI view in the form of HTML page. Finally, the participant was given a set of questionnaires to be completed and the open feedback questions to answer.

Questionnaires are used to support each part of the evaluation. The design of the questionnaires are discussed in detail in Section 9.4. Observation data is also collected while participants are performing tasks based on:

I. How they manage to complete the task given;
II. How they complete the Essential Use Cases practice manually and automatically;
III. How they navigate between different parts of the tool;
IV. How they explore the tool for consistency checking;
V. How they explore the end-to-end rapid prototyping support, and
VI. Listening to their verbal responses while using the tool.

9.2 Usability Criteria for Usability Evaluation

To evaluate our tool, we consider the type of usability criteria suggested by Lund [156] in the USE questionnaire. The author suggested four criteria that are correlated to one another- Usefulness, Ease of Use, Ease of Learning and Satisfaction. We used these criteria in developing our questionnaires, and had previously used these in our informal evaluations presented earlier. We define the criteria as follows.

- Usefulness: How useful the tool is to help users be effective in accomplishing the given task.
- Ease of Use: How easily the users can work with the tool's facility, user interface and event handler provided by the tool.
- Ease of Learning: How easily the user can understand and learn to use the tool.
- Satisfaction: Is the user satisfied with the tool's capability in solving the problems?

9.3 Cognitive Dimensions of Notations Approach (CD)

As a second element to the evaluation, we apply the CD Framework, as operationalised by Blackwell [157] in our questionnaires to allow us to explore in detail the reason for each of the user's perceptions for our MaramaAI tool in capturing requirements, managing the consistency of requirements and supporting end-to-end roundtrip prototyping. CD is applied here as it is a common approach for evaluating visual language environments. It helps non-HCI specialist and ordinary users to evaluate usability and it can be applied during any design phase [158]. In addition, it is design to provide a lightweight analysis as well as to allow reasoning about usability tradeoffs [158]. The list of CD dimensions refined by Blackwell [157] is shown in Table 9.1.

Cognitive Dimension	Meaning
Viscosity	Resistance to change
Visibility	Ability to view component easily
Premature commitment	Constraints on the order of doing
Hidden dependencies	Important links between entities are not visible
Role-expressiveness	The purpose of an entity is readily inferred
Error-proneness	The notation invites mistakes and the system gives little protection
Abstraction	Types and availability of abstraction mechanism
Secondary notation	Extra information in means other than formal syntax
Closeness of mapping	Closeness of representation to domain
Consistency	Similar semantics are expressed in similar syntactic forms
Diffuseness	Verbosity of language
Hard mental operations	High demand on cognitive resources
Progressive evaluation	Work-to-Date can be checked at any time.
Provisionality	Degree of commitment to actions and marks

Table 9. 1: CD Dimensions and Meaning by Blackwell [157]

9.4 Design of the Study

As discussed in Section 9.1, the evaluation is conducted in three different phases. Similar usability criteria and CD dimensions were evaluated for each phase. This study aimed to fulfil the following evaluation objectives:

I. to evaluate Marama AI tool's usability and effectiveness in capturing requirements, managing inconsistency and exploring the end-to-end prototyping facility, and
II. to obtain qualitative information on user perceptions of the MaramaAI tool.

We have structured our study into two parts.

1. Task list and observation

   For this part, the participants need to explore and accomplish the provided task and while they are performing the task, observation data is collected. This method aims to fulfil the second objective of the evaluation. There are two types of observation conducted.

   I. Unobtrusive observation

      Here, participants are observed on how well they use the tool. This helps us to learn whether participants can use the tool in an easy and efficient way. The following aspects are also observed.

      i) How participants capture the requirement manually and automatically and then trace the abstract interaction and map to the EUC diagram automatically.

      ii) Is a participant able to manage the consistency of the requirement?

      iii) How a participant navigates different parts of the tool and explores the facility provided for end-to-end prototyping.

   II. Obtrusive observation

      Here, participants are asked to say aloud what he/she thinks while using the tool. This helps us to learn more about the usefulness and the acceptance of the tool. Through this method, we expect the participants to feel relaxed and willing to express their sincere perception about the tool. Perceptions and comments from the participants are then collected. We took notes for each piece of feedback and each observation made.

   For both methods we are using the think aloud method [159] and no personal information about the participant is collected and no personal questions are asked.

2. Questionnaire

   Each question for each usability criteria and CD dimension was recorded using a five parts-Likert scale: 1=strongly disagree, 2=disagree, 3= undecided, 4= agree and 5=strongly agree. The results for each question blocks, which consist of several questions for each criterion, are averaged and converted to percentage. For this part, there are two sections which the participants need to answer after they have completed their tasks.

   I. The questionnaire for the four usability criteria and CD notations.

      a. For usability criteria, each criterion for part 1: capturing requirements and part 2: consistency checking and Part 3: Exploring the end to end prototyping facility (End to End Rapid Prototyping) of the evaluation are designed with three questions, with the exception that the usefulness criterion for part 3 has five questions. All these questions are designed by us with several adapted from Lund [156]. In total, our questionnaire consists of 12 questions related to the four criteria for parts 1 and 2 of the evaluation and 15 questions for part 3.

b. For CD dimensions, we do not apply all the CD dimensions provided by Blackwell in Table 1 but only focus on several elements that we think influence our tool the most and helps for better usability tradeoffs as well as better design choices discussion, which we think important for the adoption and refinement of our tool at different phases. In this evaluation, we do not consider the role-expressiveness, abstraction, secondary notation dimensions and provisionality. As for the abstraction, we think that we do not require participants to scale the level of abstraction and encapsulation provided by the tool and for role-expressiveness, we do not require the participants to discover the reasons we built the tool structure in such a way as we follows the Constantine and Lockwood methodology of creating EUC and EUI prototype models. As for secondary notation, the reason we left it out is because the notations used by MaramaAI are clearly defined and specific for a particular part of requirements. For provisionality, the reason we left it out is because at this time round, we do not require the participants to scale the degree of flexibility provided by the notations in allowing them to play with their ideas or make any marking to the design. Each CD dimension consists of one question to evaluate it. All these questions are adapted from Kutar et al. [160]. In total, there are ten questions for this section. The list of CD dimensions used by us and the questions evaluating them are shown in Table 9.2 below.

Cognitive Dimension	Question
Visibility	It is easy to see various parts of the tool
Viscosity	It is easy to make changes
Diffuseness	The notation is succinct and not long-winded
Hard mental effort	Some things do require hard mental effort
Error-proneness	It is easy to make errors or mistakes
Closeness of mapping	The notation is closely related to the result
Consistency	It is easy to tell what each part is for when reading the notation
Hidden dependencies	The dependencies are visible
Progressive evaluation	It is easy to stop and check my work so far
Premature commitment	I can work in any order I like when working with the notation

Table 9. 2: CD Notations Used and Questions Evaluating Them

Overall, there are 24 questions for part 1 and part 2 of the evaluation and there are 27 questions for part 3 of the evaluation. Two background questions are also asked at the beginning of the questionnaire regarding the participant's proficiency in using Marama tools: proficient/skilled, intermediate and novice and a question regarding his/her experience in using any tool similar to our MaramaAI.

II. Open-ended questions related to any improvements that participants' desire.

A sample of our evaluation survey appears in the appendices.

9.5 Survey Method

We invited potential participants who were enrolled in two postgraduate courses with specifically relevant background in Software Requirements, and other students who had attained a background in Software Requirements. This is because we needed participants who already had a knowledge of requirements engineering to perform this survey. We recruited 20 voluntary postgraduate students who had sufficient knowledge or experience in software requirements and requirement engineering to participate in this survey. The usability evaluation was conducted individually in order to allow us to observe participants and receive feedback one-to-one from them.

Participants were given an explanation and demonstration of how to use the prototype tool and the tasks they needed to perform. A task list and a questionnaire sheet were given to participants before they started using the prototype tool. The task list and questionnaires (Part 1, 2 ,3) as well as Consent Forms, a University Ethics Approval Form and Personal Information Sheet are in the appendices. A brief overview of the tasks for each phase of evaluation follows.

Part 1: Capturing requirements

I. Extract manually the Essential Use Cases from the scenario given.
   - The participant reads through the given scenario and extracts the abstract interaction and designs an EUC model from the scenario. The time used for this task is taken and their work checked for accuracy by comparing their answers with the EUC patterns developed by us.

II. Extract a similar scenario to the Essential Use Cases using the MaramaAI tool.
   - The participant is asked to insert the same scenario into the tool and to extract the EUC using the tool event handler.

III. Explore MaramaAI facilities in capturing requirements using event handlers: trace, trace back and map to EUC.
    - The participant is asked to explore the tool facilities in tracing forward/back and mapping using the provided event handlers.

Part 2: Consistency Checking

I. Explore the tool capability in managing the inconsistency by adding a new abstract interaction or textual requirement, delete any components or change the sequence of the components.
    - The participant is asked to explore the tool facilities for managing the consistency of requirements by doing some modifications to the requirements as instructed. Participant feedback while exploring is recorded.

II. Check for other requirements quality such as correctness and completeness using the tool.
    - The participant is asked to check their modified requirement model with a defined EUC pattern template for certain scenarios. He/she is required to observe the visual differences provided to detect the incorrectness and incompleteness in the modified requirement model. Participant feedback while exploring is recorded.

Part 3: Exploring the end-to-end prototyping facility (End-to-end Rapid Prototyping)

I. Explore the tool capability for mapping the EUC diagram to an EUI prototype.
    - The participant is asked to explore the tool facilities in supporting end-to-end rapid prototyping. He/she is asked to map the EUC model to a low-fidelity prototype: EUI prototype. Participant feedback while exploring is recorded.

II. Explore the tool facility for mapping the EUI prototype to the concrete UI in a form of HTML page.
    - The participant is asked to explore the tool facility in generating automatically the concrete UI view in a form of HTML page from the generated EUI prototype.

We observed the participants' performances while using the tool to accomplish the provided task. Participants were also asked to think aloud and give suggestions to enhance the tool. Once all tasks were completed for each part, they had to answer the questionnaire sheet provided earlier. Participants completed the questionnaire at their own pace without any supervision. The response data was then collected for analysis. Each participant took less than one hour to perform the evaluation survey. The results of the survey and analysis are discussed in the following section.

9.6 Survey Result and Analysis

In this section, we present the survey results and analysis for all three parts of the evaluation.

<u>Part 1: Capturing requirements</u>

I. Task 1: Extract manually the Essential Use Cases from the scenario given.

The accuracy results provided by all the 20 participants for this task are poor, confirming the preliminary study we undertook, presented in Chapter 3. Few participants could provide correct abstract interaction for the EUC model. Table 9.3 summarises the results of our study. The correctness (Y for correct, X for incorrect) and time taken were recorded for each person. A correct answer (Y) means that the answer provided by the participant is the same or very similar to the abstract interaction pattern developed by us following the Constantine and Lockwood [4] methodology provided in the essential interaction pattern library. Summarising these results:

4. The number of correct interactions identified (Y) = 48 out of 120 total correct interactions or 40% (i.e. 60% were incorrect).
5. The number of completely correct EUC interactions (all Ys) = 2 out of 20 or 10%.
6. The average time taken to accomplish the EUC development task was 10.2 minutes. The longest time taken was about 20 minutes and the shortest time taken was about five minutes, so there was significant variation in the time taken.

Based on these results, participants were more likely to generate incorrect EUC interactions than correct ones, and very unlikely (10%) to produce a completely correct EUC. All but two participants failed to identify some of the essential interactions present in the given requirements; many failed (highlighted in orange in Table 9.3) to assemble these into an appropriate interaction sequence, and only two (participants 6 and 20) managed to obtain a solution which is the same as or very similar to the model answer of the reserving a vehicle developed by us. From these results, it is obvious that participants took considerable time to provide the right answer. This is shown by the time taken by both participants 6 and 20 who respectively took 18 minutes and 12 minutes to provide the right answer. Our survey thus supports the preliminary findings in our initial study discussed in Chapter 3.

Participants	Answers												Time Taken (minutes)
	Choose		Offer choice		View detail		Request identification		Identify self		Confirm booking		
1.	Y		Y			X	Y			X		X	14
2		X	Y			X		X		X		X	11
3.		X		X		X		X	Y		Y		10
4.		X	Y			X		X		X	Y		10
5.		X	Y			X		X		X	Y		7
6.	**Y**		**Y**		**Y**		**Y**		**Y**		**Y**		**18**
7.		**X**		**X**		**X**		**X**		**X**		**X**	**9**
8.		X		X		X	Y		Y		Y		9
9.		**X**		**X**		**X**		**X**		**X**		**X**	**16**
10.	Y			X		X		X		X		X	6
11.		**X**		**X**		**X**		**X**		**X**		**X**	**6**
12.		**X**		**X**		**X**		**X**		**X**		**X**	**8**
13.		X	Y		Y			X		X	Y		5
14.		X	Y		Y			X		X	Y		20
15.	Y		Y		Y			X		X	Y		7
16.		X	Y		Y			X		X		X	5
17.		X	Y		Y			X		X	Y		13
18.		X	Y			X	Y		Y		Y		9
19.	Y			X	Y			X	Y		Y		9
20.	**Y**		**Y**		**Y**		**Y**		**Y**		**Y**		**12**
TOTAL	**6**	**14**	**12**	**8**	**8**	**12**	**5**	**15**	**6**	**14**	**11**	**9**	**204 =10.2**

Table 9. 3: Manual Extraction of EUC Study Result

Observation results for Task 1:

We found that participants seemed to have difficulty in finding the right level of abstraction for the abstract interactions. Most did not know how to abstract the requirements and just listed the functional requirements. We also found that it was quite time-consuming for participants to figure out appropriate keywords to describe each abstract interaction and to organise these into an appropriate sequence of user intentions and system responsibilities.

Task 2: Extract a same scenario to the Essential Use Cases using MaramaAI tool.

Task 3: Explore MaramaAI facilities in capturing requirement using event handlers: trace, trace-back and map to EUC.

Observation Results for Tasks 2 and 3:

We found that 15 out of 20 of the participants were quite happy to use the tool as most of the process was automated. They were also happy as the abstract interactions were provided and mapped to interaction sequence in the EUC model automatically by the tool. They did not have to worry about the accuracy issues observed in the first task. They also thought this would save much time and effort in capturing the essential requirements as on average, most took only 1.5 minutes to solve the task. This was much faster than the time taken in manual extraction. However, a minority group of five participants were confused and uncomfortable with the layout of Marama as well as the shape and colour used to represent the EUC model. These problems were solved after a few explanations and trials using the tool.

Background Information

As mentioned in Section 9.3, participants were asked about their proficiency in using the Marama tool and their experience in using any tool similar to our Marama AI before they moved to usability and CD notation study. The results appear in Table 9.4.

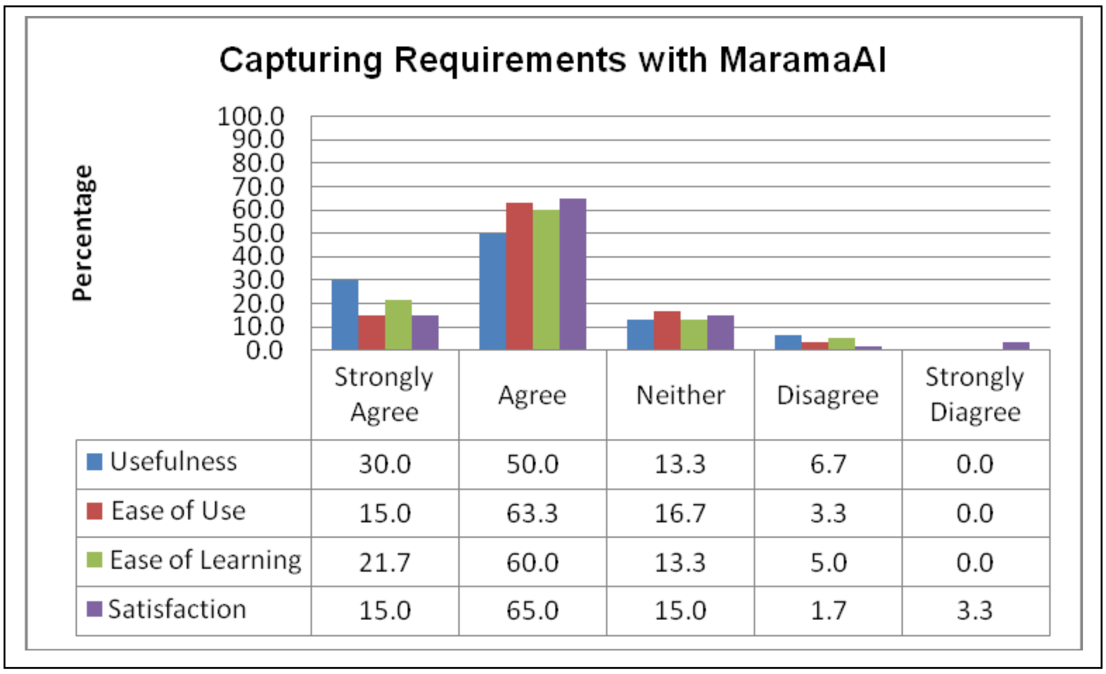
Participants	Level of proficiency in using Marama tool	Experience with any tool to capture requirements similar to Marama AI
1.	Intermediate	No
2.	Novice	No
3.	Novice	No
4.	Novice	No
5.	Novice	No
6.	Novice	No
7.	Novice	No
8.	Novice	No
9.	Novice	No
10.	Intermediate	No
11.	Novice	No
12.	Novice	No
13.	Intermediate	No
14.	Novice	No
15.	Novice	No
16.	Novice	No
17.	Intermediate	No
18.	Intermediate	No
19.	Novice	No
20.	Novice	No

Table 9. 4: Proficiency level of Using the Marama tool and Experience with Any Other Tool

Based on the background results provided in this table, our participants were novice to intermediate in using the Marama tool, the meta toolset used to construct MaramaAI – experience with this indicates they have a background in using graphical modelling toolsets and graphical tool design. Most of them were thus unfamiliar with tool design. The same group of participants was also used in our next phase of evaluation. We conclude that this group of users were unfamiliar with RE tools like MaramaAI.

Usability Criteria and CD Study

The results for the usability criteria and CD dimensions based on the questionnaire are shown in Figure 9.1 and Table 9.5.



	Strongly Agree	Agree	Neither	Disagree	Strongly Diagree
🟦 Usefulness	30.0	50.0	13.3	6.7	0.0
🟥 Ease of Use	15.0	63.3	16.7	3.3	0.0
🟩 Ease of Learning	21.7	60.0	13.3	5.0	0.0
🟪 Satisfaction	15.0	65.0	15.0	1.7	3.3

Figure 9. 1: Usability Results-Capturing Requirements

Figure 9.1 shows the survey results for each usability criterion. For each criterion, the results of each corresponding three-question block were averaged to produce the results shown. The results are positive. 80% of the participants strongly agree or agree on its usefulness in capturing requirement; the ease of use - over 78% strongly agree or agree; ease of learning - over 81% strongly agree or agree, and satisfaction (80% strongly agree or agree). For ease of use, the result is slightly lower than other criteria as a few participants felt uncomfortable with using the tool as they had difficulty in understanding the layout provided by Marama and they were used to a UML model rather than the EUC. In addition, they also expected the EUC component to be numbered in order to make it easier for users to see the sequence of interactions as, currently, the sequence is shown only as an index in the property box.

The CD study allows us to explore in more detail the reasons for these user perceptions as well as further discuss the tool's strength and weaknesses. The tradeoffs between the dimensions are also discussed in section 9.7. We used the dimensions and questions in respect to MaramaAI in Table 9.2. for this study. The results are based on percentage depending on the number of participants' answers for each scale.

Cognitive dimension	1-Strongly Disagree (%)	2-Disagree (%)	3-Neither (%)	4-Agree (%)	5-Strongly Agree (%)
Visibility	0.0	0.0	10.0	60.0	30.0
Viscosity	0.0	0.0	20.0	50.0	30.0
Diffuseness	0.0	10.0	20.0	55.0	15.0
Hard-mental effort	5.0	40.0	30.0	10.0	15.0
Error-Proneness	15.0	40.0	35.0	10.0	0.0
Closeness of Mapping	0.0	0.0	25.0	45.0	30.0
Consistency	0.0	0.0	15.0	45.0	40.0
Hidden Dependencies	0.0	5.0	15.0	60.0	20.0
Progressive Evaluation	0.0	0.0	20.0	55.0	25.0
Premature Commitment	0.0	5.0	15.0	60.0	20.0

Table 9. 5: Evaluation Results for Cognitive Dimensions Questions

Based on Table 9.5. we could summarise the results for each dimension as follows.

i. Visibility

About 90% of the participants either strongly agreed or agreed that the tool is able to show clearly the three components requirements: textual natural language requirements in the textual editor, abstract interaction and EUC in Marama Essential. They could also easily see the dependencies of each component as a visual link and highlights are provided. The remaining 10% hoped for sequence numbering for abstract interactions and the EUC components in the shapes rather than just in the property boxes.

ii. Viscosity

About 80% of the participants either strongly agreed or agreed that the tool allowed them to make changes easily. They could make changes in any part of the requirements components either in textual natural language requirement or abstract interaction or EUC model. 20% doubted the tool's ability to support independent changes as they believed that the three requirements components were dependent on one another.

iii. Diffuseness

About 70% of the participants either strongly agreed or agreed that the notation used by the tool is succinct and not long-winded. However, 10% disagreed and thought it was hard to understand the notation when using it for the first time. They were confused with the coordination, shape and colour used in representing abstract interaction and EUCs.

iv. Hard-mental effort

About 45% of the participants either strongly disagreed or disagreed that this tool needs a lot of effort to solve the tasks. They were quite happy as this tool is able to extract automatically the EUC which minimises a lot of their time and effort. This is in stark contrast to the difficulty found by users in understanding and applying EUCs found in the prior studies. However, there was still some dissatisfaction from 25% of the participants who thought this tool still required effort to understand the shape and the layout when using it for the first time. 30% of the participants answered ‘undecided’: they may have thought that all the problems faced in understanding the tool could be solved if they used it more often.

v. Error-Proneness

More than half the participants either strongly disagreed or disagreed that the tool leads the user to make errors. This is because the extracted abstract interaction is believed to be accurate as all the essential interaction and abstract interaction patterns are already pre-defined in the library. However, 35% of the participants were undecided: they may have believed that the tool could be constrained by the size of the library. Another 10% of the participants agreed that they made mistakes easily at the beginning as they were confused with the shape used for abstract interaction and the EUC model, and the highlighting of several essential interactions in the textual natural language requirements for a particular abstract interaction when trace-back was performed.

vi. Closeness of Mapping

Most participants (75%) either strongly agreed or agreed that the notation used was closely related to the results: abstract interaction and EUC model. They understood the shapes and labels used to describe both requirement components. Only 25% of the participants were undecided with the notation used as they were not familiar with the Marama meta-tool and were not happy with the colours used to identify specific shapes.

vii. Consistency

Most participants (85%) either strongly agreed or agreed that they could easily identify the requirements components: textual natural language, abstract interaction and EUC model throughout the task. Only 15% of the participants were undecided: they were unsure about the Marama shape and the colours used but believed that the notations used were consistent and straight-forward.

80% of the participants either strongly agreed or agreed that the dependencies among the three requirements components were visible. Visual links are provided to show the dependencies between abstract interaction and the EUC model when trace-back is performed. Highlights with (***) and change of colour also help to visualise the dependencies among components. However 5% of the participants disagreed with this and 15% were undecided. This may be because of the misunderstanding of the requirements components and the purpose of highlighting the essential interactions in the textual natural language requirement.

viii. Progressive Evaluation

80% of the participants also either strongly agreed or agreed that MaramaAI allows users to evaluate their work at any time and to verify the abstract interaction produced by the library. Here, participants could make any change to the list of abstract interaction if they did not agree with the tool's decision. Only 20% were undecided with this dimension. The latter is well supported by the tool as the automation process is supported by event handlers. Event handlers will only generate the event if there is a trigger from the user.

ix. Premature Commitment

This dimension reflects the sequence of using this tool in order to achieve the results. 80% of the participants strongly agreed or agreed that the tool allows a user to perform the task from any direction. However, 5% of participants disagreed and another 15% were undecided. This could be because of the constraint; only one way is provided if they want to trace the abstract interaction from the textual natural language requirement.

Open-ended Questions to Improve the Tool

The open-ended feedback for the open-ended question to improve the tool is illustrated in Table 9.6 below.

Participants	Comment
1.	"User who understands the concept of Essential Use case will be much easier to operate the tool. After regular use, it will be easy to use the tool. It will be good if the list of actions is numbered so that the user can view the sequence of interactions."
2.	"I think this tool can have better GUI for better use."
3.	"It is getting easier to use the tool if it is explored more than once."
4.	None
5.	"Re-organise the layout and the colour of the shape."
6.	"It will be better if the shape is numbered."
7.	None
8.	None
9.	None
10.	"The user interface can be improved. The boxes and the colour bar can be combined."
11.	"It will be good if the list of abstract interaction can be edited in the shape."
12.	"It is quite confusing that 3 occurrences of "indicate" keyword corresponded to one box in EUC diagram."
13.	"Better presentation of diagrammatic elements (shapes, colours) for more visual distance."
14.	"Allow tracing all together rather than individuals."
15.	None
16.	"Allow line-breaks in the text areas, solid colouring for the shapes."
17.	"A *"trace back all"* feature which would highlight all occurrences in the text might be useful and *"trace back highlights"* in the text with colour instead of stars."
18.	"Highlight the keywords in the original requirements which have been used, so user can quickly identify the situations where a requirement needs to be manually added."
19.	"Clearer differences between components (colour). The one in pink on the left almost looks like a status bar."
20.	"A larger database for the pattern is needed. A strong database could support the tool to provide accurate results."

Table 9. 6: Open-Ended Feedback

Part 2: Consistency Checking

We conducted a second part of the evaluation for the next iteration of our prototype focussing on checking the consistency. Before the evaluation was conducted, the tool was modified over a period 3- 6 weeks based on the relevant feedback and suggestions received from the first evaluation as well as extending the tool with the additional functionality developed for the iteration.

Task 1: Explore the tool capabilities for managing inconsistency by adding a new abstract interaction or textual requirement, delete any components or change the sequence of the components.

Here, participants were required to do some prescribed modifications and then observe how well the tool automatically supported the consistency validations.

Task 2: Check for other requirements quality such as correctness and completeness using the tool.

Here, participants were required to explore how the tool supports the identification of incompleteness and incorrectness that occurs in the generated EUC model.

Observation Result for Task 2 and 3:

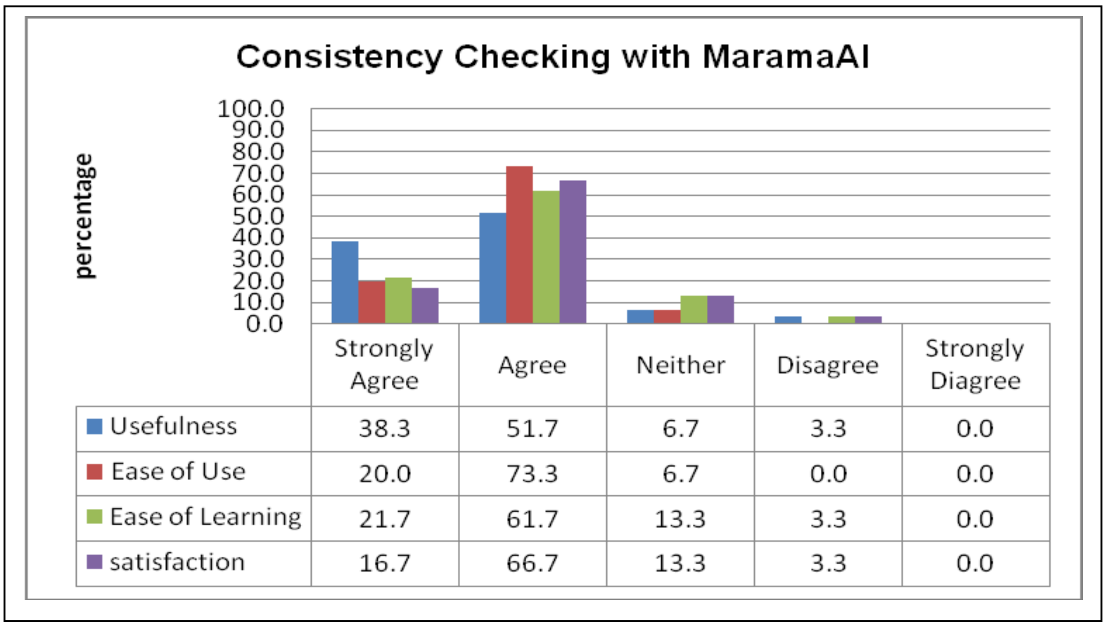
We found that 17 out of 20 participants were interested in using the tool as they were able to view automatically the inconsistencies, incompleteness and incorrectness errors that occur. They were also quite satisfied with the notification supports such as problem marker and warning, and visual differencing support in detecting the errors. However, two participants requested more modification options and validation support which focuses not only on these three types of errors. In addition, one participant strongly thought this tool would be constrained by the available templates in the EUC interaction library.

Background Information

The same group of participants from the first evaluation were used for this second evaluation. Thus the same background applies for all the participants. This is because they already had an idea of how our tool works and they could compare the current tool functionality with the previous one. Thus, we can sum up that none had prior experience with any RE tool similar to our MaramaAI for checking the consistency of requirements except for one participant who thought that the way our tool notifies the error and feedback by using warnings is similar to the Marama Critic tool. But overall, that tool has a different focus than our tool, which focuses on managing the consistency and validating the requirements, whereas Marama Critic focuses on providing a critic specification tool that allows the tool developers to construct critics for a DSVL tool [161]. Here, the results gained from the first phase of the evaluation were also used to enhance the prototype before we moved to this evaluation– note there were minor changes from the responses in Figure 9.1 and Table 9.5 due to the time difference between the two parts of the study.

Usability Criteria and CD Study

The results of the usability criteria and CD notations based on the questionnaire are shown in Figure 9.2 and Table 9.7.



	Strongly Agree	Agree	Neither	Disagree	Strongly Diagree
🟦 Usefulness	38.3	51.7	6.7	3.3	0.0
🟥 Ease of Use	20.0	73.3	6.7	0.0	0.0
🟩 Ease of Learning	21.7	61.7	13.3	3.3	0.0
🟪 satisfaction	16.7	66.7	13.3	3.3	0.0

Figure 9. 2: Usability Results - Consistency Checking of Requirements

Figure 9.2 shows the result of our usability survey on consistency checking of requirements using our tool, MaramaAI. For each criterion, the results of each corresponding three-questions block were averaged to produce the results shown. The results are very positive, with strong agreement over the usefulness of the tool (90% strongly agree or agree on its usefulness), the ease of use (over 90%), ease of learning (over 80%) and satisfaction (over 80%). These results show that there is an increment in the usefulness of the tool and other criteria compared to the previous evaluation. The small number of cases of disagreement over usefulness, ease of learning and satisfaction related to a preference by those participants for a UML Use Case-based approach rather than the Essential Use Case approach. Some also felt that requirements engineers would be too constrained by the templates available in the EUC interaction pattern library, and some could not foresee the purpose of the tool after checking the consistency. However, overall these results are very encouraging, particularly given prior studies, our own and others, that suggest EUC modelling, while appealing to end users, has a large barrier to entry due to difficulty of use [39].

The CD study allows us to explore in more detail the reason for these user perceptions. Similar to the previous evaluation, we used the dimensions and questions in Table 9.2. for this study. The results are based on percentage, depending on the number of participants' answers for each scale.

Table 9.7 shows the evaluation results for each of these questions. These demonstrate interesting tradeoffs between the dimensions that we feel have contributed to the strong usability acceptance.

Cognitive dimension	1-Strongly Disagree (%)	2-Disagree (%)	3-Neither (%)	4-Agree (%)	5-Strongly Agree (%)
Visibility	0	0	10	80	10
Viscosity	0	0	0	75	25
Diffuseness	0	0	10	70	20
Hard-mental effort	5	40	45	10	0
Error-Proneness	0	55	45	0	0
Closeness of Mapping	0	5	15	85	5
Consistency	0	0	15	80	5
Hidden Dependencies	0	0	15	70	15
Progressive Evaluation	0	0	15	50	35
Premature Commitment	0	0	15	65	20

Table 9. 7: Evaluation Result for Cognitive Dimensions Questions

Based on Table 9.7, we summarise the results for each dimension as below;

i. Visibility

90% of the participants strongly agree or agree that they could see various parts of the tool. They could easily view the consistency dependencies among the three requirement components: textual natural language requirements, abstract interaction and EUC model. Only 10% of the participants are undecided.

ii. Viscosity

All participants strongly agree or agree that they find it easy to make changes to the diagrams representing the various notational forms. Both strong results of visibility and viscosity show that the participants are comfortable with the tool.

iii. Diffuseness

About 90% of the participants strongly agree or agree that the notation used by the tool is succinct and not long-winded. However, 10% of the participants are undecided with this as they are more comfortable with UML diagrams than with the EUC model.

iv. Hard-mental effort

About 45% of the participants strongly disagree or disagree that this tool needs a lot of effort to solve the tasks. They are quite happy as this tool is able to automatically detect inconsistencies in the requirements. However, there is still dissatisfaction from 10% of the participants who think this tool still requires effort and strong understanding to locate the error if they do not understand the concept of the EUC model. 45% of the participants answer undecided: perhaps they used this tool for the first time and all doubts were resolved after a few explanations and trials.

v. Error-Proneness

More than half of the participants disagree that the tool leads the user to errors. This is because all the errors are detected automatically and they could view the incorrectness and incompleteness of the differences between the generated EUC model and the templates. In addition, all modifications are also checked with the available patterns and templates in the library, so that they do not worry about the accuracy issue. Another 45% of the participants are undecided; perhaps they think that the tool might constrain the available patterns and templates available in the library. However, they believe that this problem could be easily solved when we mentioned that the collection of patterns and templates for both our libraries are on-going.

vi. Closeness of Mapping

Most participants (90%) strongly agree or agree that the notations used are relatively intuitive and understandable. Only 15% of the participants are undecided and another 5% disagree with the notations used as they are confused with the Marama Layout.

vii. Consistency

Most participants (85%) strongly agree or agree that they could easily recognise the notations used by our tool. Only 15% of the participants are undecided as they were initially confused with the notations used to represent the differences between the generated EUC model and the EUC model from the templates. However, those doubts were resolved after a few explanations.

viii. Hidden dependencies

85% of the participants strongly agree and agree that the dependencies among the three requirements components are visible. A visual link is provided to show the dependencies between abstract interaction and EUC model when inconsistencies exist. Highlights with (***) in textual natural language requirements, change of colour, problem marker as well as warning and feedback also help to visualise the inconsistencies among components when any requirement component is changed. However, 15% of the participants are undecided.

ix. Progressive Evaluation

85% of the participants also strongly agree or agree that they could easily stop and check their work at any time. MaramaAI allows changes to be made to any of the requirement components. Thus, end users do not have to worry about the errors as the tool provides an automated support if any errors such as inconsistencies, incompleteness and incorrectness exist. Only 15% are undecided but their doubts were resolved after a few trials.

x. Premature Commitment

This dimension reflects the sequence of using this tool in order to achieve the results. Over 85% of the participants strongly agree or agree that the tool allows a user to perform the task from any direction. End users could make changes in any of the components as the tool provides automated detection if the changes cause an inconsistency in any other component.

If inconsistency is detected, end users could make changes to other components as well, based on the feedback provided by the tool in order to keep the three requirements components consistent. Only 15% of the participants are undecided but their doubts were resolved after a few trials.

Open-ended Questions to Improve the Tool

The open-ended feedback for the open-ended question to improve the tool is illustrated in Table 9.8.

Participants	Comment
1.	"The main concern is the pattern and template available in the library. More templates available are better."
2.	None
3.	"Allow to use templates only partially, e.g. use some parts of it but not all."
4.	"To consider more options/types of requirement quality error apart from inconsistency, incompleteness and incorrectness."
5.	"Still confuse with the layout. Need to change the colour used by the notations."
6.	None
7.	"Modification options can be increased."
8.	None
9.	None
10.	"I like the visual differencing approach as I could view visually the errors."
11.	"Easy to make changes to the notations."
12.	"Still prefer the UML rather than EUC".
13.	"Need to have knowledge and understanding on the concept of EUC."
14.	"Need to have a try on this tool for a few times."
15.	None
16.	None
17.	None
18.	"Would be too constrained by the available templates available in the library."
19.	None
20.	"More templates in the library are needed to support the visual differences."

Table 9. 8: Open-ended Feedback

Part 3: Exploring the end-to-end prototyping facility (End-to-End Rapid Prototyping)

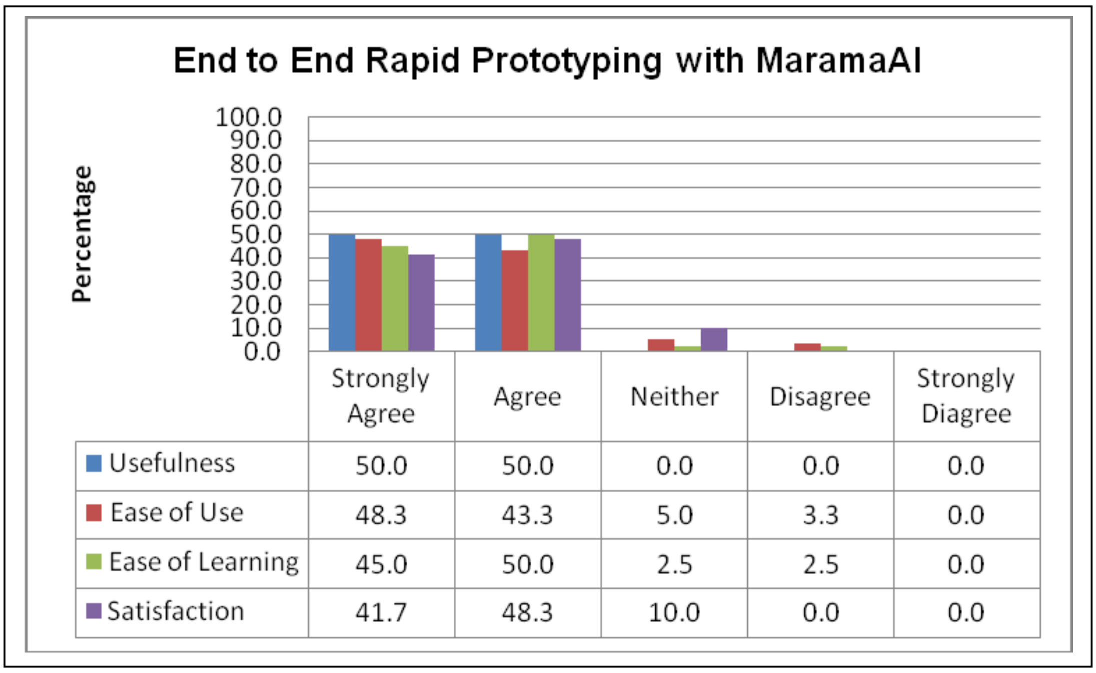
We have conducted another evaluation for the final iteration of our prototype for end-to-end rapid prototyping support. Before the evaluation was conducted, the tool was again modified over a period of 3-6 weeks based on the relevant feedback and suggestions received from the second evaluation, as well as adding additional functionality.

Task 1: Explore the tool capability in mapping the EUC diagram to the EUI prototype
Here, the participants were required to explore the tool utility in mapping the generated EUC model to the low-fidelity prototype-EUI prototype.
Task 2: Explore the tool facility for mapping the EUI prototype to the concrete UI in a form of HTML.
Here, the participants were required to explore the tool facility to translate the EUI prototype model to the more concrete UI view: an HTML form. From here, participants could see the end results and confirm the consistency of the requirements provided earlier in a form of textual natural language requirements with the generated prototype.

Observation Results for Tasks 2 and 3:
We found that 16 of the 20 participants were happy to use the tool as they were able to view the prototype as a result from the generated EUC model. They were also satisfied with the generation of the concrete UI view–HTML form as they could understand and have the picture of the requirements instead of just text and model. However, one participant requested GUI support for the HTML form based on particular domains and three other participants suggested that a video demo be embedded in the tool. Those three participants also requested more descriptive labels and a key to explain the meanings of the colours near or at the side of the models.

Background Information
The same group of participants from the first and second evaluations was used in this third evaluation, so the same background applies. This is because they already had an idea of how our tool works and what the tool supports and they could compare the usefulness of the current tool with the previous one. We also concluded that no participant has experienced the same approach of end-to-end rapid user interaction prototyping provided by MaramaAI in any other requirements tool in capturing or analysing the requirements.

Usability Criteria and CD Study
The results of the usability criteria and CD notations based on the questionnaire are shown in Figure 9.3 and Table 9.9.



	Strongly Agree	Agree	Neither	Disagree	Strongly Diagree
🟦 Usefulness	50.0	50.0	0.0	0.0	0.0
🟥 Ease of Use	48.3	43.3	5.0	3.3	0.0
🟩 Ease of Learning	45.0	50.0	2.5	2.5	0.0
🟪 Satisfaction	41.7	48.3	10.0	0.0	0.0

Figure 9. 3: Usability Results- End to End Rapid Prototyping

Figure 9.3 shows the results of the usability survey conducted for the end-to-end rapid prototyping approach provided by our tool. For each characteristic, the results of each corresponding question block were averaged to produce the results shown. The results are very positive compared to the previous evaluation, with strong agreement over the usefulness of the tool (100% strongly agree or agree on its usefulness), the ease of use (over 90%), ease of learning (95%) and satisfaction (90%). The small number of cases of disagreement over ease of use and ease of learning related to a preference by those participants to have a more descriptive label for each colour and shape used in MaramaAI.

The CD study allows us to explore in more detail the reasons for these user perceptions. Similar to previous evaluations, we used the dimensions and questions in Table 9.2 for this study. The results are based on percentage, depending on the number of participants' answers for each scale. Table 9.11 shows the evaluation results for each of these questions. As with the previous evaluation, we believe these results demonstrate interesting usability tradeoffs between the dimensions that we feel have contributed to the strong usability acceptance of our final prototype.

Cognitive dimension	1-Strongly Disagree (%)	2-Disagree (%)	3-Neither (%)	4-Agree (%)	5-Strongly Agree (%)
Visibility	0.0	0.0	5.0	50.0	45.0
Viscosity	0.0	0.0	5.0	50.0	45.0
Diffuseness	0.0	0.0	5.0	55.0	40.0
Hard-mental effort	30.0	50.0	15.0	5.0	0.0
Error-Proneness	25.0	50.0	10.0	15.0	0.0
Closeness of Mapping	0.0	0.0	0.0	40.0	60.0
Consistency	0.0	5.0	10.0	55.0	30.0
Hidden Dependencies	0.0	0.0	10.0	50.0	40.0
Progressive Evaluation	0.0	0.0	5.0	50.0	45.0
Premature Commitment	0.0	0.0	5.0	40.0	55.0

Table 9. 9: Evaluation Result for Cognitive Dimensions Questions

Based on Table 9.9, we conclude that almost all results are positively increased compared to both evaluations conducted earlier. A summary of the results for each dimension follows.

i. Visibility

Almost all of the participants (95%) strongly agree or agree they could see various parts of the tool. They could easily view the dependencies between requirements components in MaramaEssential and the prototype in MaramaEUI. Only 5% are undecided as they are not sure of the notation used for the EUI prototype at the beginning.

ii. Viscosity

95% of the participants strongly agree or agree that they found it easy to make changes to the diagrams either in the EUC model or EUI prototype model. Strong results of both visibility and viscosity show that the participants were comfortable with the tool.

iii. Diffuseness

95% of the participants strongly agree or agree that the notation used by the tool is succinct and not long-winded, although there were suggestions to be able to generate HTML forms with GUI templates based on a selected (i.e. domain specific) application domain. 5% of the participants were undecided with this as they were new to the EUI prototype model.

iv. Hard-mental effort

About 80% of the participants strongly disagree or disagree that this tool needs a lot of effort to solve the tasks. They are happy as this tool is able to help to generate automatically the EUI prototype from the EUC model and to translate automatically the EUI prototype model to a more concrete UI view: HTML form. Only 5% believe that they need a lot of effort to understand the concept and dependencies between the EUC model and EUI prototype. 15% of the participants answered undecided: perhaps they used this tool for the first time and all doubts were satisfied after explanations and trials. However, this is still in strong contrast to the findings of prior studies regarding the difficulty found by users in understanding and applying EUCs and EUI prototypes.

v. Error-Proneness

75% of the participants disagree that the tool leads the user to errors. This is because all the errors are detected automatically and they could automatically generate the prototype. The prototype generated is based on the pre-defined template and UI pattern in the library. Thus, this assures the accuracy of the UI. Only 15% of the participants disagree with this and another 10% are undecided. However, all doubts were resolved after explanations were made.

vi. Closeness of Mapping

All participants strongly agree or agree that the notations used are relatively intuitive and understandable.

vii. Consistency

Most participants (85%) strongly agree or agree that they could easily recognise the notations used by our tool. Only 5% of the participants disagree with this and 10% of the participants are undecided as they were initially confused with the notations used to represent the EUI prototype model. All the misunderstandings were resolved after a few explanations.

viii. Hidden dependencies

90% of the participants strongly agree or agree that the dependencies among the three requirements components and the prototype are visible. The highlights with a change of colour to red in the EUC component and abstract interaction as well as the highlight with (**) in textual natural language requirements after trace-back are performed from the EUI prototype and show the dependencies between all requirements components and the prototype. Only 10% of the participants are undecided as some required that highlights be provided to active elements in order to view the consistency between the EUI prototype and EUC model. In addition, a warning is also requested if any EUI prototype is deleted: this ensures consistency with the EUC model.

ix. Progressive Evaluation

95% of the participants also strongly agree or agree that they could easily stop and check their work at any time. Marama AI allows changes to be made to any of the requirement components and the prototype. Only 5% are undecided but all doubts were resolved after a few explanations.

x. Premature Commitment

This dimension reflects the sequence of using this tool in order to achieve the results. Over 95% of the participants strongly agree or agree that the tool allows a user to perform the task from any direction. End users could make changes in any of the components either from the MaramaEUI editor or Marama Essential editor. End users could use either or both editors to capture the requirements. Only 5% of the participants are undecided but they still believe they could use the tool in any direction they like.

Open-ended Questions to Improve the Tool

The open-ended feedback to the open-ended question to improve the tool is illustrated in Table 9.10 below.

Participants	Comment
1.	"Highlight active elements to view the consistency between EUI prototype and EUC models. Provide warning if any item of EUI prototype is deleted."
2.	None
3.	"Remove unnecessary elements from eclipse. Provide more descriptive labels."
4.	"Could provide templates for specific domain for the HTML form."
5.	"EUC and EUI prototype can be on the same page."
6.	None
7.	None
8.	None
9.	None
10.	"Would prefer to have a video demo on how to use the tool ."
11.	"Maybe could have a "key" explaining what the colours mean near to the model."
12.	"Would be good if the HTML form is designed with interesting GUI."
13.	"Need to understand the concept of EUI prototype."
14.	"Need to have a try on this tool for a few times."
15.	None
16.	None
17.	None
18.	None
19.	None
20.	"MaramaAI tool can be improved by providing more interactions between abstract interaction and other diagrams. In future, it can be used to extract UML diagram such as class diagram and sequence diagram."

Table 9. 10: Open-ended Feedback

9.7 Comparison of Survey Results

We compared the results gained from these three evaluations. Most results showed a positive increment in terms of usability, especially the usefulness of the tool. The comparison results of the usability study for the three evaluations- capturing requirements, consistency checking and end-to-end rapid prototyping are shown in Figure 9.4.

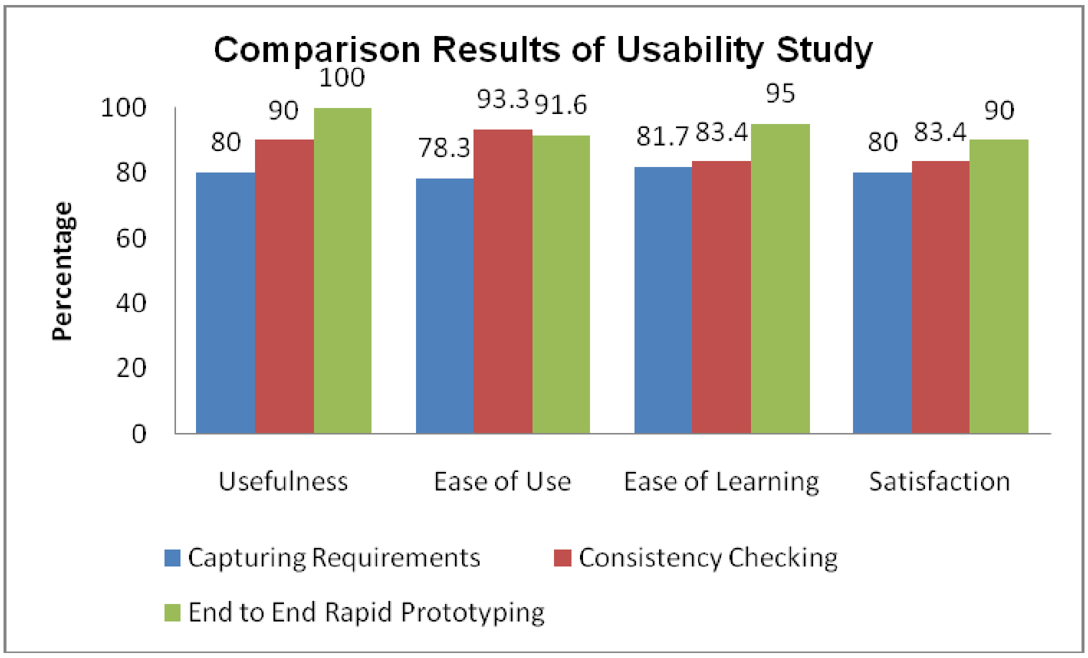


Figure 9. 4: Comparison Results of Usability Study for Capturing Requirements, Consistency Checking and End-to-End Rapid Prototyping of MaramaAI

Based on Figure 9.4, the usefulness of the tool increases in each of the evaluations. All participants agreed that the final prototype, end-to-end rapid prototyping, is useful. They could understand the value of the tool for consistency management and validation of requirements after they viewed the results of the requirements in terms of the prototype.

Other results of usability criteria also show positive increment for each of the prototype iterations. For ease of use, from only 78.3% rating in the first prototype (capturing requirements), increases to 93.3% in the second prototype (consistency checking), but has a slight drop with 91.6% in the final prototype. This is because most of the participants are still new to the concept of abstract prototype (EUI prototype) and the tool is somewhat more complex with the additional functionality provided.

However, for ease of learning criteria, the rating of the final prototype increases to 95% from just 83.4% in the second prototype and 81.7% in the first prototype. This is because most participants believe that they could easily follow the flow of the tool after a few trials and explanations. The satisfaction rate also gains a positive increase in the final prototype with 90% compared to only 83.4% in the first prototype and 80% in the second prototype. Most participants are satisfied with the end results produced as the tool visualises the requirements in a form of prototype.

As we used a CD study to explore in more detail the reasons for these user perceptions, we compared the results for each dimension for the three evaluations. We also explored the tradeoffs between the dimensions. There are also positive increments for each of the dimensions in the CD study. The comparison results for each dimension in the CD study are shown in Figure 9.5.

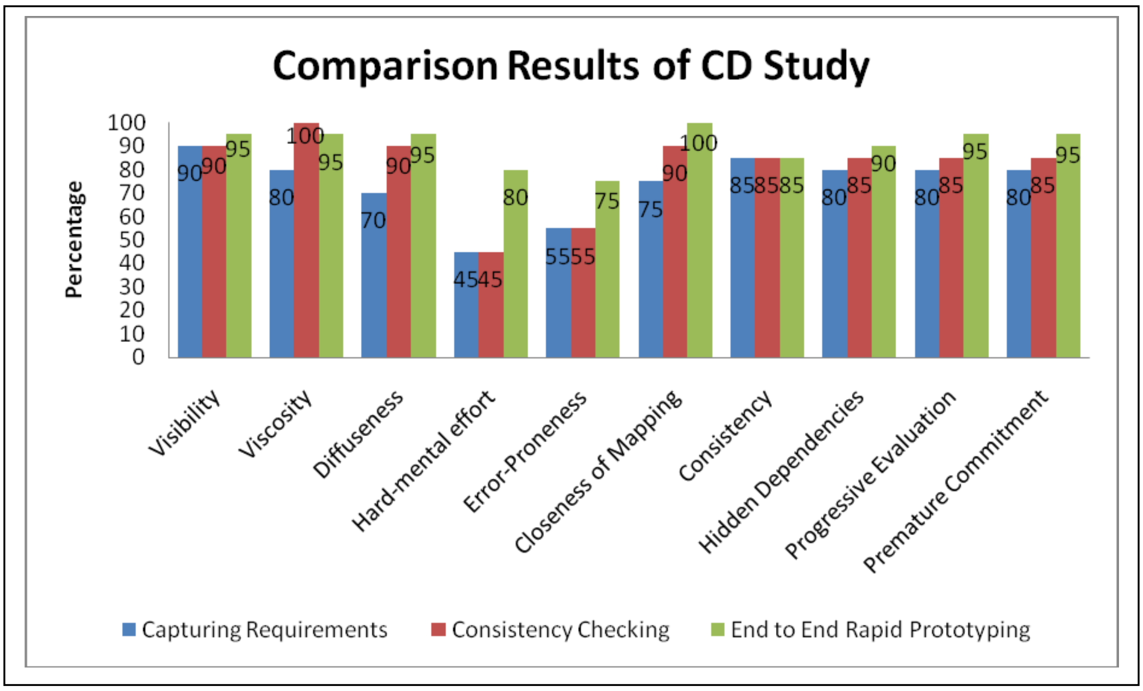


Figure 9. 5: Comparison results of CD Study for Capturing Requirements, Consistency Checking and End-to- End Rapid Prototyping of MaramaAI

Figure 9.5 shows that most of the results for each of the CD dimensions have increased in each of the prototype iteration. The visibility rating has increased to 95% in the final prototype compared to 90% in both the first and second prototypes.

The viscosity rating has a high increment from 80% in the first prototype, shooting up to 100% in the second prototype. It shows that the participants began to be comfortable with the tool after they have understood the concept of an EUC model. However, there is a slight drop to 95% in the final prototype as most participants are still new to the concept of the EUI prototype.

The results of diffuseness also show a high increment, with only 70% in the first prototype, increasing to 90% in the second prototype and 95% in the final prototype. This shows that most participants start to understand and be comfortable with the notation used in MaramaAI.

For the hard mental effort dimension, only 45% of the participants in both first and second prototypes agreed that the tool is easy to use, but the rate increases to 80% in the final prototype. This high increment also happens in the error-proneness dimension, where 75% of the participants of the final evaluation compare favourably to both earlier prototypes where only 45% agree that it is hard to make errors or mistakes with the tool. The result for closeness of mapping also increased to 100% in the final prototype compared to 90% in the second prototype and only 75% in the first. The strength of mapping rating and the relatively strong hard mental effort and error proneness ratings indicate that EUC and EUI prototype notations are seen as intuitive and understandable by the participants. This is in stark contrast to the difficulty found by users in understanding and applying EUC's and EUI prototype in the prior studies.

The result of consistency dimension for the three evaluations remains at 85%. The result for the hidden dependencies dimension also shows gradual increments for each prototype. The first prototype received an 80% rating and the second prototype with an 85% rating while the final prototype had a 90% rating. These results show that participants are improving in understanding the dependencies that occur between the requirements components and the prototype.

A similar gradual increment also happens to the rating of progressive evaluation and premature commitment dimensions. For both dimensions, the first prototype received an 80% rating and the second prototype had a 85% rating and the final prototype had a 95% rating. We conclude that the automated extraction support, consistency management and generation of prototypes appear to be responsible for all these high ratings.

From the results, the usability tradeoffs between the dimensions are also clearly shown. It shows that high closeness of mapping and visibility as well as high viscosity assists with the issue of hard mental operations and reduces the error proneness. The high progressive evaluation also contributes to the lower error- proneness and hard mental operations. The high visibility also increases the result of hidden dependencies. The consistent result of consistency affects the increment of the diffuseness result. In addition, it is also shown that the high premature commitment assist with the positive result of the high progressive evaluation.

9.8 Summary

The MaramaAI tool has been evaluated phase by phase with a usability survey and Cognitive Dimensions study. The evaluation results are positive for all the three phases of prototypes: capturing requirements, consistency checking and end-to-end rapid prototyping.

The survey has shown very positive results in usefulness. This shows a good degree of acceptance by end-users to use the tool in managing the consistency and validating the requirements. The results appear to complement the prior studies in applying EUCs and the EUI prototype.

However, there are also some minor improvements are needed to improve the usability of the MaramaAI tool. The evaluation survey has also provided a number of suggestions (listed in the previous sections) to improve the usability of MaramaAI. The suggestions will be taken into consideration in our future work.

Chapter 10: Conclusion and Future Work

This chapter concludes this thesis by summarising the contributions of this research responding to research questions described in Chapter 1 as well as some conclusions on the achievements so far. This chapter also discusses the limitations of the research and suggests some future work to extend the research.

10.1 Summary of Research Contributions

1. We have designed and developed a lightweight extraction approach to deal with natural language requirements. This lightweight approach is implemented in an automated tracing tool which provides facilities for authoring textual natural language requirements and checking the consistency of those requirements. This approach enables requirements engineers to extract quickly and accurately essential requirements (abstract interactions) from the textual natural language requirements and then map them to an Essential Use Case (EUC) model. To support the extraction process, we have also developed an essential interaction pattern library and a collection of abstract interaction patterns and essential interactions patterns which are reusable and can be applied in various domains of applications.

2. We have designed and developed requirements analysis support to validate the requirements' consistency and quality. We have implemented automated traceability and visualisation support to manage the consistency of the requirements in three different forms: textual natural language requirements, abstract interactions and EUC models; as well as to further validate the correctness and completeness of requirements. To do this, we employed a visual differencing approach between essential interaction patterns and EUC interaction patterns. Thus, this could assist the requirements engineers to find appropriate interactions for designing the EUC model for a particular system. Warnings and highlights are also used to highlight inconsistencies and other requirement quality errors such as incorrectness and incompleteness.

3. We have designed and developed a rapid prototyping approach together with traceability support to provide end-to-end support for consistency checking which we assume will be usable by both requirements engineers and clients to confirm the consistency of requirements. In addition, an approach has been developed and embedded into the tool to automatically map the semi-formal requirements in the form of EUC model to an abstract

Essential User Interface (EUI) prototype model and a more concrete UI view in a form of a HTML page. The traceability support provides trace forward and trace- back between the EUI prototype, EUC model, abstract interaction and textual natural language requirements. We have also developed a set of EUI patterns to enhance the accuracy of a generated EUI prototype model and a set of EUI pattern templates to allow the translation of an EUI prototype model to the more concrete UI view – HTML page for various domains of applications.

4. We have developed a proof of concept prototype tool: MaramaAI. This was built using the existing Marama meta-tool and acts as a proof of concept for our approaches for providing automated support for consistency management and validation of requirements. We have evaluated our prototype tools performance and efficacy mainly for the tool extraction process and we have evaluated the tool with an end-user study, confirming the usability of the tool based on Cognitive Dimensions (CD) and applied the tool to several case studies in different domains of applications.

10.2 Conclusions

From our research, we conclude that our automated support tool,MaramaAI, can extract automatically abstract interactions and EUC models from textual natural language requirements. Then, an EUI prototype model and concrete UI view in the form of an HTML page can also automatically be generated from the EUC model. We have demonstrated that these automation processes perform better than manual processes conducted by requirements engineers.

In addition, our tool can also check for the 3 C's - consistency, correctness and completeness - using the developed essential interaction patterns and EUC interaction patterns with traceability and visual support to highlight the requirements' quality errors such as inconsistencies, incorrectness and incompleteness.

Finally, we have also speculated that our approach and automated tool support are able to improve the dialogue between the requirement engineers and stakeholders/clients by having the auto-generated prototypes from the EUC models: these help to provide a clearer picture of the requirements to the client. It also allows confirmation of the consistency of the requirements captured by the engineers with a client's original requirements by having the textual natural language requirements, abstract interaction, EUC model, EUI prototype model and HTML form mutually traceable to each other.

However, some limitations of our MaramaAI tool were exposed by the evaluations and the case studies applications. These include the following.

1. Constraints on the size of the essential interaction pattern library, specifically the list of abstract interactions and essential interactions available. During the evaluations, the tool efficacy result averaged 80% due to grammar issues and the size of the library. This is also supported by the example of the case study three in Chapter 8, in which we also faced the problem of tracing a particular abstract interaction. However, this can be readily addressed as the process of abstracting and storing patterns is on-going.
2. Constraints on the available types of "best-practice" templates in the EUC interaction patterns library for supporting the visual differences in validating requirements. More "best-practice" templates are required to support wider domains of applications. However, this also can be handled as the process of searching for and storing the templates is on-going.
3. Constraints on the available EUI pattern and template in the EUI pattern library for generating the prototypes: EUI prototype and HTML page. More EUI patterns and a EUI pattern templates are also needed to generate more UI for wider domains of applications. In addition, generation of prototype concrete UIs that are not form based may be desirable.
4. Constraints in handling validation of multiple requirements. At present, the validation process can only be conducted subsequently by MaramaAI and the tool does not allow partial acceptance of a particular EUC interaction pattern from the "best-practice" template.
5. Constraints on a novice user to understand the layout of the tool. During the evaluation, participants requested better representation including the layout, colour, shape and labels used by the tool. A video demo on how to use the tool and illustration explaining the colours and shapes was also requested.
6. The HTML page generated does not contain any GUI. It only shows the main functionality of the requirements.

Overall, these minor limitations observed in our tool can be improved in future work.

10.3 Future Work

MaramaAI tool is still at the prototype stage. We aim to continually develop it to be more robust and to allow its use by the Requirement Engineering community and industry.

In order to achieve this, we need to address the following matters.

1. Enhance the support of our library for essential interaction patterns, EUC interaction patterns and EUI patterns. To do this, a pattern template editor can be developed to allow automatic updates of the patterns to be done by any requirements engineer. At present, the process of updating the patterns is via manual insertion by the developer into the libraries. With appropriate editing support, a requirements engineer could update or define any new abstract interaction, essential interaction, EUC interaction or EUI pattern in the various libraries following the provided guidelines.

2. Provide better support for consistency management and validation of multiple requirements. We plan to enhance the support tool to allow validation for all requirements at the same time. With this, the consistency checking can be done for all requirements at the same time. The completeness and correctness checking using the visual differences also can be conducted together for all requirements. In addition, we also plan to provide the user with flexibility either to change the whole or partially of original EUC interaction against the "best-practice" template.

3. Enhance the layout of MaramaAI and embed a video demo describing how to use the tool for better understanding by a novice user. The colour and shapes used in the tool need some improvement with better labelling to explain the features. The tool can also be integrated with a GUI template for the generated HTML form for each domain of applications. The GUI template, based on different domains of applications, can be stored in a library to allow automatic generation of the template together with the generated HTML form.

4. Conduct an evaluation of the effectiveness of our tool in improving the dialogue between the requirements engineer and the stakeholders/clients. This evaluation is to assess the effectiveness of our tool-MaramaAI utilities in assisting the communication between the requirement engineer and clients, especially in confirming the consistency of requirements captured by the requirements engineer using the tool with the client's original requirements. We are now waiting for Ethics approval for this evaluation, which will be underway shortly..

5. Integrate our tool with other UML models such as sequence diagrams and class diagrams to check for the consistency of the generated EUC model with both UML sequence and class diagram. The next step is to try to integrate our framework with other NLP tools in order to perform grammar analysis for the textual natural language requirements in the textual editor. With this, we may overcome some of the grammar problem faced by our current tool.

10.4 Summary

The research presented here has focused on providing automated support for consistency management and validation of requirements. We have developed several libraries of patterns, covering essential interactions, EUC interactions and EUIs which, together with traceability and visual support for the EUC concept, we have demonstrated to be usable, for managing consistency and validating requirements. In addition, the dialogue between the requirements engineer and the client may also be improved via the generation of user interface prototypes from the EUC model. Evaluations focussing on the performance and efficacy of the tool extraction process and formal evaluations on the tool usability as well as a demonstration of the use of the tool with different case studies were employed to evaluate the approach and the proof of concept prototype tool - MaramaAI.

APPENDIX A: PARTICIPANT INFORMATION SHEET (HEAD OF DEPARTMENT)

THE UNIVERSITY OF AUCKLAND
FACULTY OF ENGINEERING
Department of Electrical and Computer Engineering

Science Centre (Building 303)
38 Princess Street
Auckland, New Zealand
Telephone 6493737599 ext 88158
Facsimile 6493737461
www.auckland.ac.nz

The University of Auckland
Private Bag 92601
Auckland 1142
New Zealand

PARTICIPANT INFORMATION SHEET (HEAD OF DEPARTMENT)

Title: Evaluation of Marama AI: Automated Inconsistency Checker for Capturing Requirement and Managing Inconsistency

My name is Massila Kamalrudin and I am a PhD student at the Department of Electrical and Computer Engineering, The University of Auckland. I am conducting research on Inconsistency checking of Software Requirements. This research is under the supervision of Professor John Hosking and Professor John Grundy. Our research investigates the use of an automated tool which applies an approach using visualisation and lightweight traceability techniques for capturing and formalising natural language requirements and managing consistency between the natural language and formalised requirements in an efficient and simple way. A prototype tool for capturing requirement and managing inconsistency called *Marama AI* has been developed. Part of our research involves an evaluation of this prototype regarding its usability and effectiveness for capturing natural language requirement and managing the inconsistency and exploring the end to end prototyping facility that is provided by the tool. As a Computer Science Head of Department, we would like to ask your permission to allow us to have access to students who enrolled in COMPSCI 732 course and SOFTENG 450 course, or any postgraduate or undergraduate student who has a background of Software requirement and permit the students to participate voluntarily in our survey. Participation in this survey is on a voluntary basis and there will be no financial compensation. The survey is performed in an anonymous way. No personal information will be collected during the survey. A Participant Information Sheet (PIS) and consent form will be given to students before they start with the evaluation process in order to make sure they understand the terms and conditions. Once, they understand and agree with both documents and they wish to continue participation, they will need to sign the consent form. Both documents will be collected immediately after they agree to participate in the evaluation and before they start with the evaluation. A tutor from their class will collect the questionnaires once they have completed the evaluation and answered all the questionnaires". We would like you to provide us the assurance that neither the students' grades nor academic relationships with the department staff members will be affected by either refusal or agreement in students' participation. Your support would be greatly appreciated.

This research is funded by the Ministry of Higher Education, Malaysia. If you have any queries regarding this survey, please do not hesitate to contact me. You can email me at: mkam032@aucklanduni.ac.nz. Alternatively, you may phone me at 0210 -2446787. You may also contact my supervisor, Professor John Hosking at john@cs.auckland.ac.nz or 09 373 7599 ext 88297.

For any queries regarding ethical concerns you may contact the Chair, The University of Auckland Human Participants Ethics Committee, The University of Auckland, Office of the Vice Chancellor, Private Bag 92019, Auckland 1142. Telephone 09 373-7599 extn. 83711.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE ON 6 May 2010 for (3) years, Reference Number 2010/172.

APPENDIX B: PARTICIPANT INFORMATION SHEET (STUDENT)

THE UNIVERSITY OF AUCKLAND
FACULTY OF ENGINEERING
Department of Electrical and Computer Engineering

Science Centre (Building 303)
38 Princess Street
Auckland, New Zealand
Telephone 6493737599 ext 88158
Facsimile 6493737461
www.auckland.ac.nz

The University of Auckland
Private Bag 92601
Auckland 1142
New Zealand

PARTICIPANT INFORMATION SHEET (STUDENT)

Title: Evaluation of Marama AI: Automated Inconsistency Checker for Capturing Requirement and Managing Inconsistency

My name is Massila Kamalrudin and I am a PhD student at the Department of Electrical and Computer Engineering, The University of Auckland. I am conducting research on Inconsistency checking of Software Requirements. This research is under the supervision of Professor John Hosking and Professor John Grundy. Our research investigates the use of an automated tool which applies an approach using visualisation and lightweight traceability techniques for capturing and formalising natural language requirements and managing consistency between the natural language and formalised requirements in an efficient and simple way. A prototype tool for capturing requirement and managing inconsistency called *Marama AI* has been developed. Part of our research involves an evaluation of this prototype regarding its usability and effectiveness for capturing natural language requirement and managing the inconsistency and exploring the end to end prototyping facility that is provided by the tool.

You are invited to participate in this survey as you are either postgraduate or undergraduate student who enrolled COMPSCI 732 course or 4th year undergraduate student who enrolled SOFTENG 450 course, or any postgraduate or undergraduate student who has a background of Software requirement. Your comments and assistance would be greatly appreciated.

Participation in this survey is on a voluntary basis and there will be no financial compensation. The survey is performed in an anonymous way. No personal information will be collected during the survey. You can be assured that neither your grades nor academic relationships with the department staff members will be affected by either refusal or agreement to participate. This assurance is given by the Computer Science Head of Department. You can withdraw yourself from the survey at any time. Completing the required tasks in the survey and submitting the evaluation is an indication of consent but as the evaluation is anonymous, no answers can be withdrawn once the evaluation is submitted.

If you consent to participate in this survey, the participation involves one visit to the Computer Science Undergraduate Laboratory, approximately 1 hour. A Participant Information Sheet (PIS) (this document) and consent form will be given to you before you start the evaluation process in order to make sure you understand the terms and conditions. Once you understand and agree with both documents and you wish to continue participation, you will need to sign the consent form. Both documents will be collected immediately after you agree to participate in the evaluation and before you start with the evaluation. You will be given an explanation together with a demonstration of what need to be done. A task list and questionnaire sheet will be given to you before you start using the prototype tool. You will be asked to extract Essential Use cases manually, to perform a number of tasks on the prototype tool and once you completed the task, you will be asked to answer the questionnaire sheet given to you.
A tutor from you class will collect the questionnaires once you have completed the evaluation and answered all the questionnaires. You also will be observed to allow the researcher to learn whether the tool is easy and efficient to use and also to know more about the usefulness and acceptance of the tool. You will be observed based on the following aspects: a) how you manage to complete the task given to you; b) how you complete the Essential Use Cases practice manually and automatically; c) how you navigate different parts of the tool; d) how you explore the tool for consistency checking and end-to end rapid prototyping and d) your verbal responses while using the tool. The observations will take place only while you perform the tasks on the prototype tool. There will be note-taking while you perform the tasks and also while you are responding or commenting when using the prototype tool. However, no personal information will be collected in this observation process. Audio-tape, video-tape and any other electronic means such as Digital Voice Recorders are not used in this survey.

After completing the tasks you will be asked to answer the questionnaire sheet. Once you completed the questionnaire, you need to put in the box that will be placed in the lab. There will be no coding to your questionnaire as it is treated anonymously. The observation and questionnaires data will be compiled and analysed, and the results will be used for a PhD thesis and for other academic publications. Results also will be available to participants on request. The observation and questionnaires data will be stored for SIX (6) years for the purpose of peer review and further research. When the observation and questionnaires data is no longer needed, it will be destroyed using the paper shredder.

This research is funded by the Ministry of Higher Education, Malaysia. If you have any queries regarding this survey, please do not hesitate to contact me. You can email me at: mkam032@aucklanduni.ac.nz. Alternatively, you may phone me at 0210 -24426787. You may also contact my supervisor, Professor John

Hosking at john@cs.auckland.ac.nz or 09 373 7599 ext 88297, or the Head of Department, Professor Gill Dobbie, gill@cs.auckland.ac.nz or 09 373 7599 ext 83949, or you can write to us at:

Department of Computer Science
The University of Auckland
Private Bag 92019
Auckland.

For any queries regarding ethical concerns you may contact the Chair, The University of Auckland Human Participants Ethics Committee, The University of Auckland, Office of the Vice Chancellor, Private Bag 92019, Auckland 1142. Telephone 09 373-7599 extn. 83711.

APPROVED BY THE UNIVERSITY OF AUCKLAND HUMAN PARTICIPANTS ETHICS COMMITTEE ON 6 May 2010 for (3) years, Reference Number 2010/172

APPENDIX C: CONSENT FORM (HEAD OF DEPARTMENT)

Department of Computer Science
Level 3, Science Centre
Building 303
38 Princes St
The University of Auckland
Private Bag 92019
Auckland

Tel: 09 373 7599

<u>CONSENT FORM (HEAD OF DEPARTMENT)</u>

This Consent Form will be held for a period of six (6) years.

Title: *Evaluation of Marama AI: Automated Inconsistency Checker for Capturing Requirement and Managing Inconsistency*

Researcher: Massila Kamalrudin

I have read and understood the Participant Information Sheet. I understand the nature of the research and why I have been asked for permission and assurance of this research. I understand that this research is funded by Ministry of Higher Education Malaysia. I have had the opportunity to ask questions and have them answered. I agree to support the survey.

- I agree to allow the researcher to have access to the students who enrolled in COMPSCI 732 course and SOFTENG 450 course, or any postgraduate or undergraduate student who has a background of Software requirement.
- I agree to permit the students to participate voluntarily in the survey.
- I understand there will be no payment to the student who participates in the survey.
- I understand that all of the data collected from the survey will be non-identifying.
- I agree to provide the assurance that neither grades nor academic relationship with any departmental staff members will be affected by either refusal or agreement to students' participation in the survey.

Name:___________________________________

Signature & Date: _________________________

APPENDIX D: CONSENT FORM (STUDENT)

Department of Computer Science
Level 3, Science Centre
Building 303
38 Princes St
The University of Auckland
Private Bag 92019
Auckland

Tel: 09 373 7599

CONSENT FORM (STUDENT)

This Consent Form will be held for a period of six (6) years.

Title: *Evaluation of Marama AI: Automated Inconsistency Checker for Capturing Requirement and Managing Inconsistency*

Researcher: Massila Kamalrudin

I have read and understood the Participant Information Sheet. I understand the nature of the research and why I have been selected to participate in this research. I understand that this research is funded by Ministry of Higher Education Malaysia. I have had the opportunity to ask questions and have them answered. I understand that I can withdraw at any time but that data already recorded cannot be withdrawn. I agree to take part in the survey.

- I understand that I will not be paid for the time taken to participate in this survey.
- I understand that all of the data collected from the survey will be non-identifying.
- I understand that I will be observed while doing a task on the prototype tool if I agree to participate in this survey. No audio-tape, video-tape or any other electronic means such as Digital Voice Recorders is used in this survey.
- I understand that I will need to fill in a questionnaire at the end of the task if I agree to participate in this survey.
- I understand that only the researcher and her main supervisor will have access to the questionnaire and observation data.
- I understand that the observation and questionnaire data may be used to review the research outcomes both to improve the notation and software tool and in publications about the survey.
- I understand that data will be archived or stored for six years and then destroyed.
- I understand that the Computer Science Head of Department has provided assurance that neither my grades nor academic relationship with any department staff members will be affected by either refusal or agreement to participate.
- I understand that at the conclusion of the survey, a summary of the results will be available from the researcher upon request.

Name:__________________________________

Signature & Date: _________________________

APPENDIX E: SURVEY QUESTIONNAIRES

Department of Computer Science
Level 3, Science Centre
Building 303
38 Princes St
The University of Auckland
Private Bag 92019
Auckland

Tel: 09 373 7599

Survey: Evaluation of Marama AI: Automated Inconsistency Checker for Capturing Requirements and Managing Inconsistency

Note: This survey is structured into THREE parts. Part one, provides a scenario that need to be extracted to an Essential Use Cases diagram (EUC) by you, manually and then try it in an automated way using the Marama AI. An observation data will be collected by PhD student Massila Kamalrudin while you are performing these tasks. You are provided with a set of questionnaire that should be answered by you once participant have completed the tasks.

Statement

	I have read the Participant Information Sheet and have understood the nature of the survey and I agree to take part in this survey. (please tick √)

PART ONE: Capturing Requirements

Purpose: To allow you to gain experience in capturing software requirements from natural language scenarios using an Essential Use Cases (EUC) diagram manually and automated. After designing the diagram manually, the participant needs to try to repeat the task using the Marama AI: Automated Inconsistency Checker tool. The same scenarios are applied to the tool. Please note that you will be observed on how you extracted the requirements manually and how you used the tool. You can ask questions while doing the task. Observation data will be collected while participant carry out this task.

Instruction: Please read and perform the following task steps.

Task 1: Understanding the Essential Use Case (EUC) Modelling Approach

What is an Essential Use case?

Essential Use Cases (EUCs) are part of Usage-Centered Design, as developed by L. Constantine and L.Lockwood [1]. The authors defined an essential use case as:

"An essential Use Case is a **structured narrative,** expressed in the language of the application domain and of users, **comprising simplified, generalized, abstract, technology free and implementation independent description** of one task or interaction that is complete, meaningful, and well-defined from the point of view of users in some role or roles in relation to a system and that embodies the purpose or intention underlying the interaction."

EUCs are documented in a format representing a dialogue between the user and the system that is user intention and system responsibility. The labels indicate how EUC support abstraction by allowing interaction to be documented without describing, the details of the user interface. The **abstraction does not really relate to the use case as a whole, but**

more to the steps of the use case. In this way, an EUC does not specify a sequence of interaction, but a **sequence with abstract steps**.

The EUC:

User intention	**System responsibility**
1. Identify self	2. Verify identity

Example of use case step for "getting cash"[2]:

1. The use case begins when the Client insert an ATM card[1]. The system reads and validates the information[2] on the card.
2. System prompts for pin. The client enters PIN[1]. The system validates the PIN[2].
3. System asks which operation[3] the client wishes to perform. Client selects "Cash withdrawal[4]."
4. System request amounts. Client enters amount.
5. System request type. Client selects account type[4] (checking, saving, credits)
6. The system communicates with the ATM network to validate account ID, PIN and availability of the amount requested.
7. The system asks the client whether he or she wants receipt. This step is performed only if there is paper left to print the receipt.
8. System asks the client to withdraw the card. Client withdraws card. (This is security measure to ensure that clients do not leave their cards in the machine.)
9. System dispenses the requested amount [5] of cash.
10. System prints receipt.
11. Client receives cash[6].
12. The use case ends.

User intention	**System responsibility**
1. Identify self	2. Verify identity
	3. Offer choices
4. choose	5. dispense cash
6. take cash	

Task 2. Extract natural language requirements and design an Essential Use Cases (EUC) model from these

Scenario 1:

Time Taken:	**(minutes)**

Reserve a vehicle:

1. This use case begins when a customer indicates he wishes to make a reservation for a rental car.

2. The system prompts the customer for the pickup and return locations of the reservation, as well as the pickup and return dates and times. The customer indicates the desired locations and dates.

3. The system prompts for the type of vehicle the customer desires. The customer indicates the vehicle type.

4. The system presents all matching vehicles available at the pickup location for the selected date and time. If the customer requests detailed information on a particular vehicle, the system presents this information to the customer.

5. If the customer selects a vehicle for rental, the system prompts for information identifying the customer (full name, telephone number, email address for confirmation, etc.). The customer provides the required information.

6. The system presents information on protection products (such as damage waiver, personal accident insurance) and asks the customer to accept or decline each product. The customer indicates his/her choices.

7. If the customer indicates "accept reservation," the system informs the customer that the reservation has been completed, and presents the customer with a reservation confirmation.

8. This use case ends when the reservation confirmation has been presented to the customer.

User Intention	System responsibility

Task 3. Explore Marama AI: Automated Inconsistency Checker

1. Marama AI is divided into three parts; Input textual requirements (1), List of Abstract Interactions (2) and EUC Diagram (3) called Marama Essential. You just need to insert the natural language requirement in (1) and save. Then, right-click in the diagram window and use the menu item (**Trace**) to trace the abstract interaction (2). Then, you can use the menu item (**MaptoEUC**) to map the abstract interaction to Marama Essential (3). You can use the menu item (**trace back**) if you would like to support tracing back to the requirements either from abstract interaction to textual requirement or Marama Essential to abstract interactions and textual requirements.

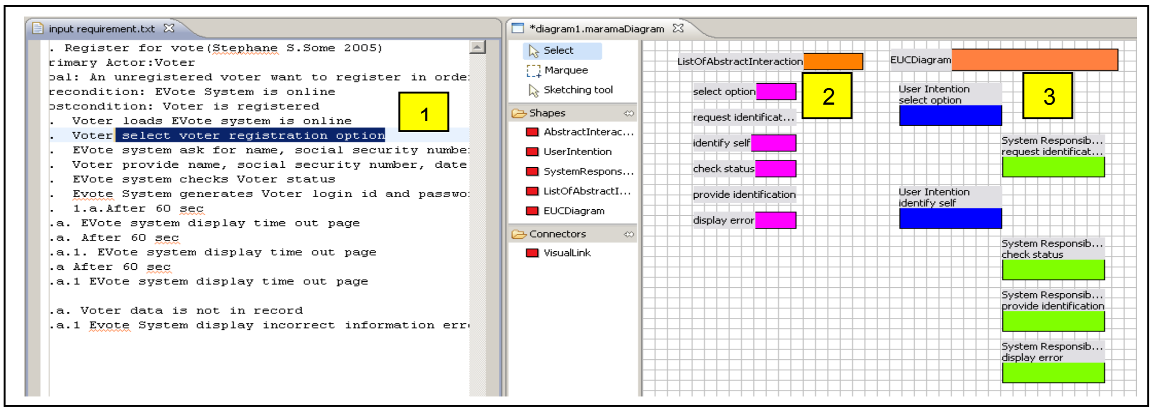

```
input requirement.txt
. Register for vote(Stephane S.Some 2005)
rimary Actor:Voter
oal: An unregistered voter want to register in orde
recondition: EVote System is online
ostcondition: Voter is registered
.  Voter loads EVote system is online
.  Voter select voter registration option
.  EVote system ask for name, social security numbe
.  Voter provide name, social security number, date
.  EVote system checks Voter status
.  Evote System generates Voter login id and passwo
.  1.a.After 60 sec
.a. EVote system display time out page
.a. After 60 sec
.a.1. EVote system display time out page
.a After 60 sec
.a.1 EVote system display time out page

.a. Voter data is not in record
.a.1 Evote System display incorrect information err
```

Figure 1.Capturing requirements with Marama AI

How to use:
1.Insert requirement in (1) and save .

2.Right click and choose Trace.

3.Right click and choose MapToEUC

4. Right click and choose trace back (optional)

References:

[1] J. Noble. R. Biddle, E.Tempero, "From Essential Use Cases to Objects," C. L. ltd, Ed.: Ampersand Press, 2002.

[2] L. A. D. Lockwood. L. L. Constantine, "Structure and style in use cases for user interface design ": Addison -Wesley, 2001.

Insert this requirement to the textual input component:

1. This use case begins when a customer indicates he wishes to make a reservation for a rental car.

2. The system prompts the customer for the pickup and return locations of the reservation, as well as the pickup and return dates and times. The customer indicates the desired locations and dates.

3. The system prompts for the type of vehicle the customer desires. The customer indicates the vehicle type.

4. The system presents all matching vehicles available at the pickup location for the selected date and time. If the customer requests detailed information on a particular vehicle, the system presents this information to the customer.

5. If the customer selects a vehicle for rental, the system prompts for information identifying the customer (full name, telephone number, email address for confirmation, etc.). The customer provides the required information.

6. The system presents information on protection products (such as damage waiver, personal accident insurance) and asks the customer to accept or decline each product. The customer indicates his/her choices.

7. If the customer indicates "accept reservation," the system informs the customer that the reservation has been completed, and presents the customer with a reservation confirmation.

This use case ends when the reservation confirmation has been presented to the customer.

Task 4: Questionnaire

Instruction:
Please answer the following questions.

Section (1)- Background Information.

1. How do you rate yourself in using Marama? (tick one box)

	Proficient/skilled
	Intermediate
	Novice

2. Have you experience with any tool that enables you to capture requirements similar to the of Marama AI?

Section (2)- Prototype Tool Information.

Please rate your agreement with the following statements about how you feel in general when using ***Marama AI:Automated inconsistency Checker*** (a new automated tool to manage requirements and check inconsistency). Please circle or tick the level of agreement that applies using the following scale:

1: Strongly Disagree (SD) **2**:Disagree (D) **3**: Undecided (U) **4**: Agree (A) **5**: Strongly Agree(SA)

A. Usefulness:
It is useful to capture the essential requirement (abstract interaction).
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It helps me be more effective in capturing requirements.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It makes it easier to capture requirements.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

B. Ease of Use:
It is easy to use.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is user friendly.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I don’t notice any inconsistencies as I use the tool.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

C. Ease of Learning:
I learned to use it quickly.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I easily remember how to use it.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to learn to use it.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

D. Satisfaction:
I am satisfied with it.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I would recommend it to a friend.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is fun to use.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

E. Cognitive Dimensions of Marama AI:

It is easy to see various parts of the tool.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to make changes.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The notation is succinct and not long-winded.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

Some things do require hard mental effort.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to make errors or mistakes.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The notation is closely related to the result.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to tell what each part is for when reading the notation.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The dependencies are visible.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to stop and check my work so far.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I can work in any order I like when working with the notation.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

F. After completing this questionnaire, can you think of obvious ways that the Marama AI tool can be improved for capturing the requirement? What are they?

____

____

____

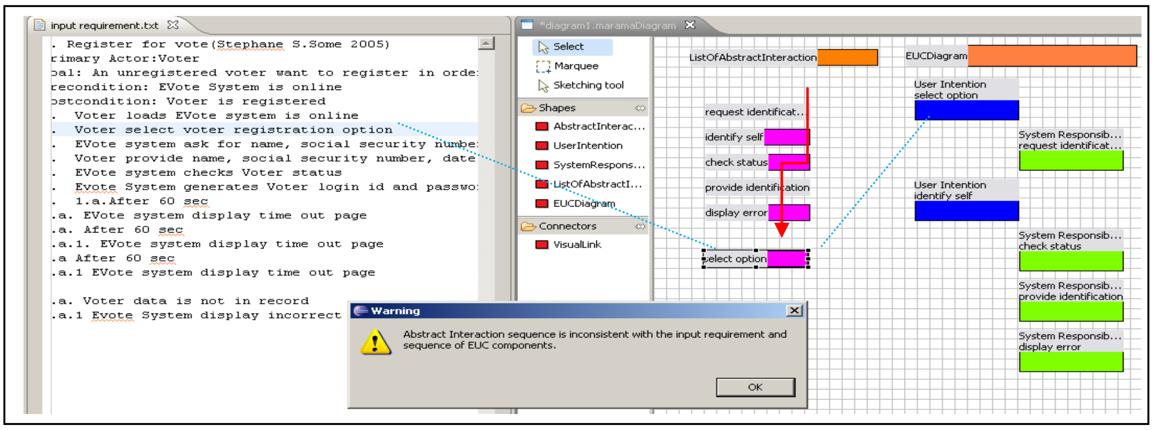
____

Thank you for your time!

Please let us know if you have any queries about this questionnaire or the survey we are conducting. Questions or concerns can either be directed to the researcher, Massila Kamalrudin(mkam032@aucklanduni.ac.nz) or to her supervisor, Professor John Hosking (john@cs.auckland.ac.nz), Dept. of Computer Science.

PART TWO: Consistency checking

Task 1:
Please explore the tool capability in managing the consistency of the requirements. There are three forms of requirements; textual, a set of abstract interactions and an EUC diagram is then checked for consistency when any of the components is changed using the inconsistency management support that is provided by the tool as shown in Figure 1.



```
. Register for vote(Stephane S.Some 2005)
rimary Actor:Voter
oal: An unregistered voter want to register in orde
recondition: EVote System is online
ostcondition: Voter is registered
.  Voter loads EVote system is online
.  Voter select voter registration option
.  EVote system ask for name, social security numbe
.  Voter provide name, social security number, date
.  EVote system checks Voter status
.  Evote System generates Voter login id and passwo
.  1.a.After 60 sec
.a. EVote system display time out page
.a. After 60 sec
.a.1. EVote system display time out page
.a After 60 sec
.a.1 EVote system display time out page

.a. Voter data is not in record
.a.1 Evote System display incorrect
```

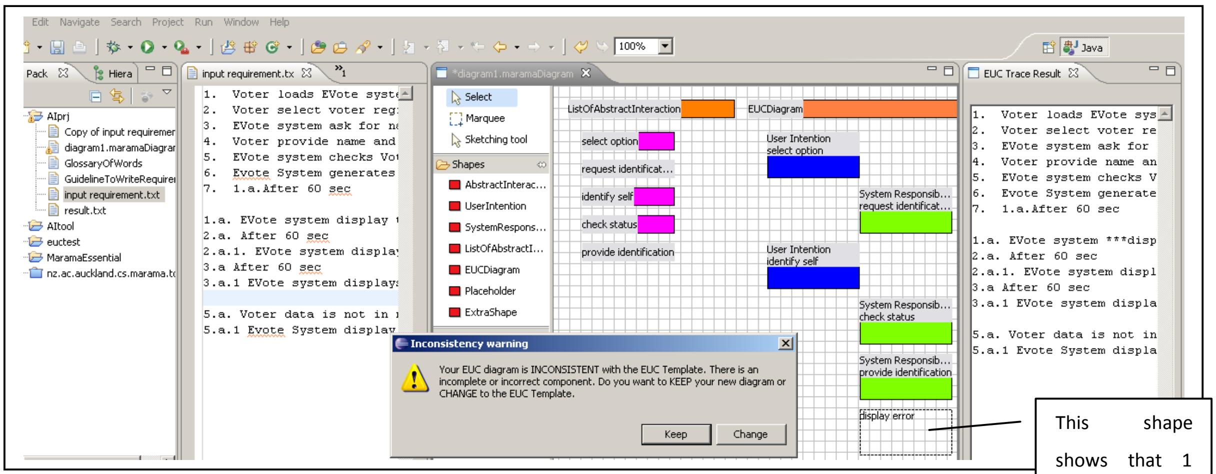
Warning

Abstract Interaction sequence is inconsistent with the input requirement and sequence of EUC components.

Figure 1.0 example of consistency checking

You can explore the tool capability in managing the inconsistency by adding a new abstract interaction or textual requirement, delete any components or change the sequence of the components. All processes are automated and use drag and drop. You can also modify the layout back to original by using the command **Reset layout.**

You can also check for the requirements quality such as correctness and completeness using the tool.
Use the command **Check consistency with a template** in order to check for the EUC diagram completeness and correctness. The result is as shown below;



```
1.   Voter loads EVote syste
2.   Voter select voter reg
3.   EVote system ask for na
4.   Voter provide name and
5.   EVote system checks Vot
6.   Evote System generates
7.   1.a.After 60 sec

1.a. EVote system display t
2.a. After 60 sec
2.a.1. EVote system display
3.a After 60 sec
3.a.1 EVote system displays

5.a. Voter data is not in r
5.a.1 Evote System display
```

EUC Trace Result

```
1.   Voter loads EVote sys
2.   Voter select voter re
3.   EVote system ask for
4.   Voter provide name an
5.   EVote system checks V
6.   Evote System generate
7.   1.a.After 60 sec

1.a. EVote system ***disp
2.a. After 60 sec
2.a.1. EVote system displ
3.a After 60 sec
3.a.1 EVote system displa

5.a. Voter data is not in
5.a.1 Evote System displa
```

Inconsistency warning

Your EUC diagram is INCONSISTENT with the EUC Template. There is an incomplete or incorrect component. Do you want to KEEP your new diagram or CHANGE to the EUC Template.

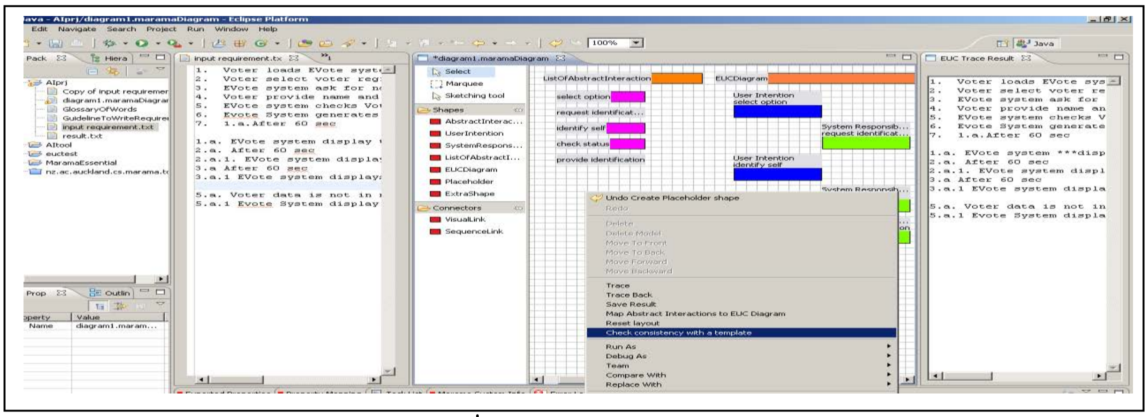
This shape shows that 1 component is missing

Figure 2.0 example of quality checking

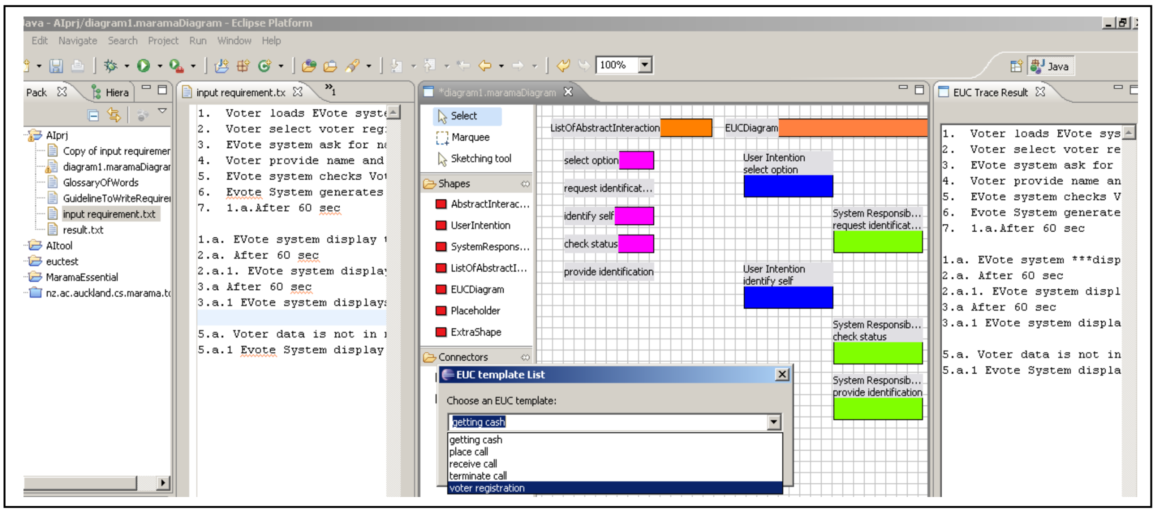
You can choose either to keep your new diagram or change the diagram based on the Template.

The step to check using the template is as below;

1) Choose the command Check consistency with a template


Java - AIprj/diagram1.maramaDiagram - Eclipse Platform
Undo Create Placeholder shape
Redo
Delete
Delete Model
Move To Front
Move To Back
Move Forward
Move Backward
Trace
Trace Back
Save Result
Map Abstract Interactions to EUC Diagram
Reset layout
Check consistency with a template
Run As
Debug As
Team
Compare With
Replace With

2) Choose the appropriate template list based on the requirement



Task 2: Questionnaire

Instruction:
Please answer the following questions.

Section (1)- Background Information.

3. How do you rate yourself in using Marama? (tick one box)

	Proficient/skilled
	Intermediate
	Novice

4. Have you experience in any tool that enables you to manage the consistency similar to the way of Marama AI?

Section (2)- Prototype Tool Information.

Please rate your agreement with the following statements about how you feel in general when using ***Marama AI:Automated inconsistency Checker*** (a new automated tool to manage requirements and check inconsistency). Please circle or tick the level of agreement that applies using the following scale:

1: Strongly Disagree (SD) **2**:Disagree (D) **3**: Undecided (U) **4**: Agree (A) **5**: Strongly Agree(SA)

G. Usefulness:
It is useful in checking the requirement inconsistency.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It helps me be more effective in managing the requirement inconsistency.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It makes the management of requirement inconsistency easier to achieve.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

H. Ease of Use:
It is easy to use.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is user friendly.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I don't notice any inconsistencies as I used the tool.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I. Ease of Learning:
I learned to use it quickly.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I easily remembered how to use it.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to learn to use it.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

J. Satisfaction:
I am satisfied with it.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I would recommend it to a friend.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is fun to use.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

K. Cognitive Dimensions of Marama AI for consistency checking:

It is easy to see various parts of the tool.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to make changes.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The notation is succinct and not long-winded.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

Some things do require hard mental effort.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to make errors or mistakes.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The notation is closely related to the result.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to tell what each part is for when reading the notation.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The dependencies are visible.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to stop and check my work so far.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I can work in any order I like when working with the notation.
Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

L. After completing this questionnaire, can you think of obvious ways that the Marama AI tool can be improved for checking the requirement inconsistency? What are they?

____

____

____

____

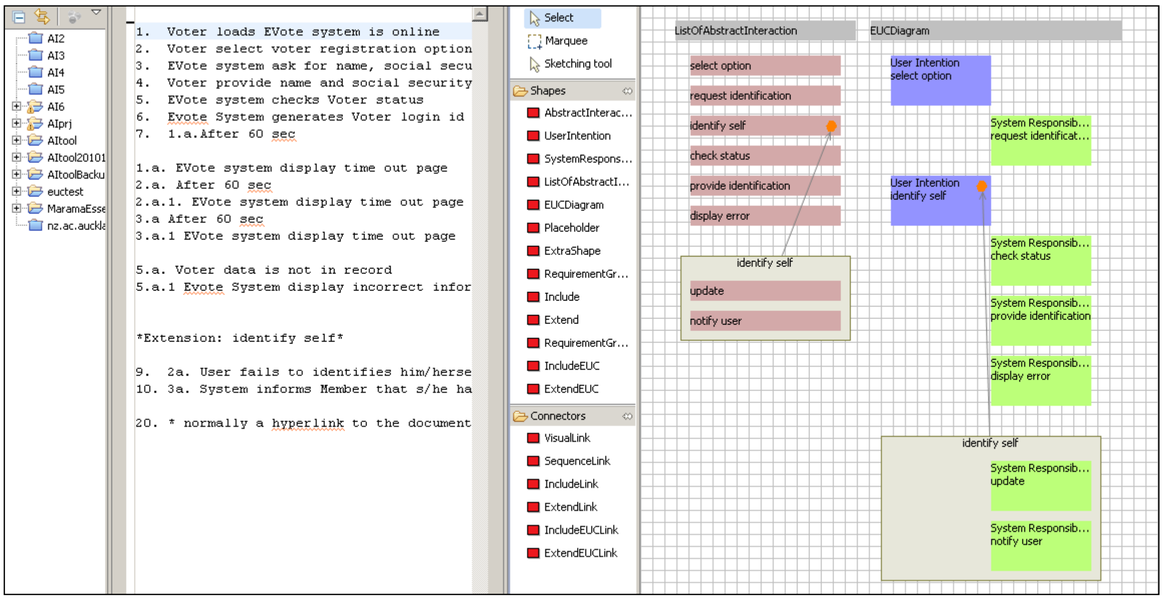
Thank you for your time!

Please let us know if you have any queries about this questionnaire or the survey we are conducting. Questions or concerns can either be directed to the researcher, Massila Kamalrudin(mkam032@aucklanduni.ac.nz) or to her supervisor, Professor John Hosking (john@cs.auckland.ac.nz), Dept. of Computer Science.

PART THREE: End to End Prototyping

TASK 1:

Please explore the refined tool capability for managing the requirements together with an end to end prototyping facility. There are three forms of requirements; textual, a set of abstract interactions and an EUC diagram. You can map these requirements to a low fidelity UI, called an Essential User Interface prototype (EUI) or Abstract Prototype. You can then map the EUI to concrete UI in the form of form based UI by using the facilities provided by the tool. Suppose you have textual, abstract interaction and EUC requirements as shown in Figure 1.



```
1.   Voter loads EVote system is online
2.   Voter select voter registration option
3.   EVote system ask for name, social secu
4.   Voter provide name and social security
5.   EVote system checks Voter status
6.   Evote System generates Voter login id
7.   1.a.After 60 sec

1.a. EVote system display time out page
2.a. After 60 sec
2.a.1. EVote system display time out page
3.a After 60 sec
3.a.1 EVote system display time out page

5.a. Voter data is not in record
5.a.1 Evote System display incorrect infor


*Extension: identify self*

9.   2a. User fails to identifies him/herse
10. 3a. System informs Member that s/he ha

20. * normally a hyperlink to the document
```

Figure1. Three components of requirement after refinement: Textual requirement, Abstract interaction and EUC diagram

You can explore the tool capability in mapping the EUC diagram (item 1 Figure 2) to the Essential User Interface (EUI) or Abstract prototype (item 2 Figure 2) by using the event handler "map EUC to EUI".

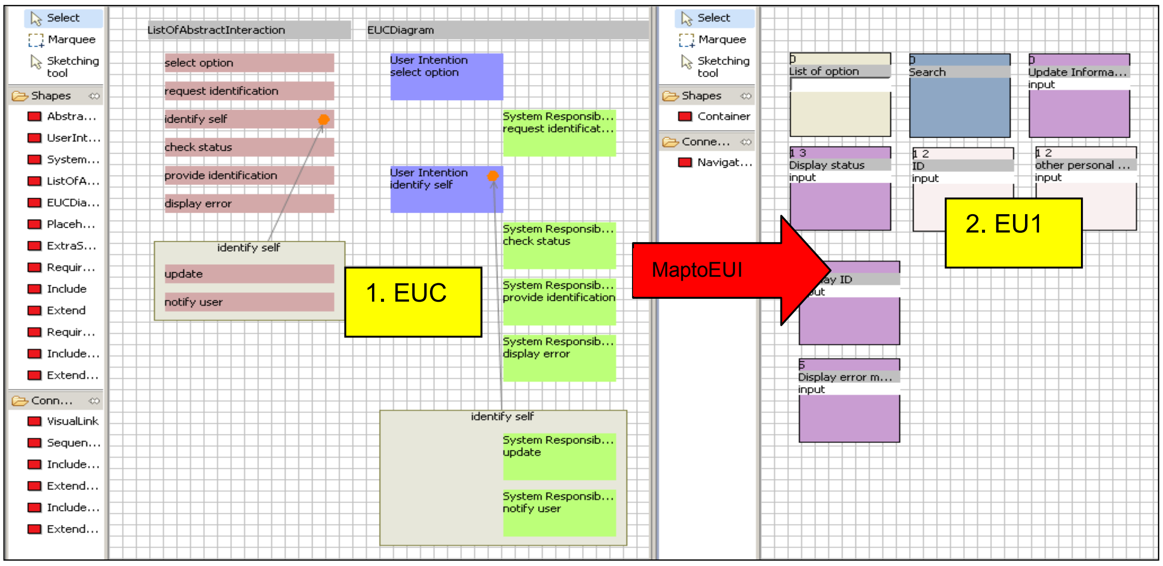
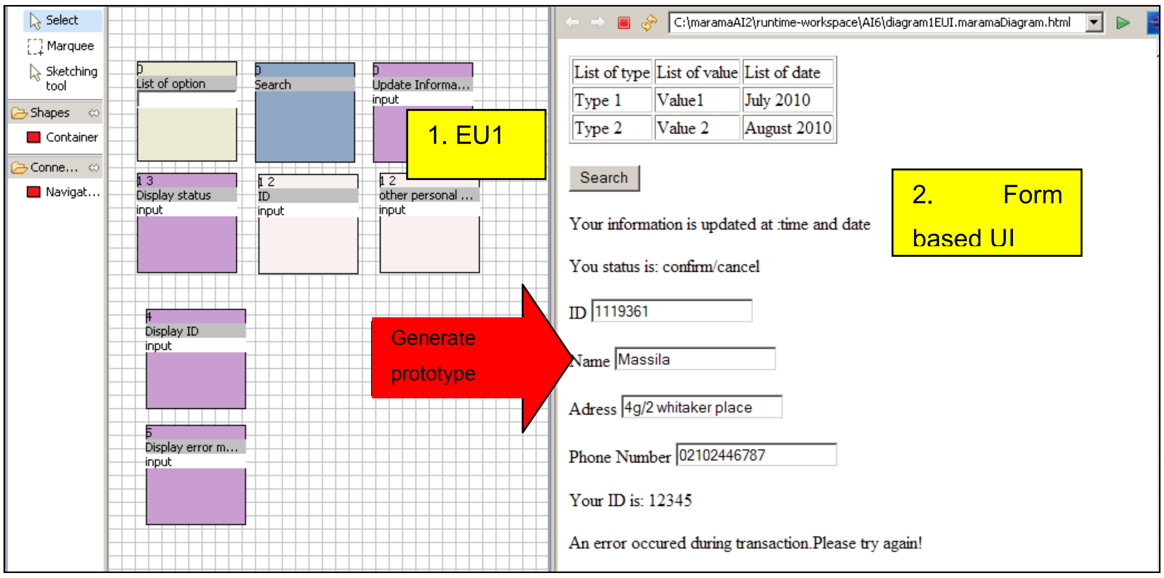


Figure 2. Mapping EUC diagram to EUI using the event handler "Map toEUI"

You can further explore the tool facility for mapping the Abstract prototype (item 1 Figure 3) to the concrete UI in a form of form based UI (item 2 Figure 3) in order to view the outcome of the models. You can also use both types of UI to confirm and verify the requirements captured from the earlier stages.



C:\maramaAI2\runtime-workspace\AI6\diagram1EUI.maramaDiagram.html

List of type	List of value	List of date
Type 1	Value1	July 2010
Type 2	Value 2	August 2010

Search

Your information is updated at :time and date

You status is: confirm/cancel

ID 1119361

Name Massila

Adress 4g/2 whitaker place

Phone Number 02102446787

Your ID is: 12345

An error occured during transaction.Please try again!

Figure 3. Mapping EUI diagram to Concrete UI-Form based UI using the event handler "generate prototype"

Task 2: Questionnaire

Instruction:

Please answer the following questions.

Section (1)- Background Information.

5. How do you rate yourself in using Marama? (tick one box)

	Proficient/skilled
	Intermediate
	Novice

6. Have you experience with any tool that enables you to apply end to end rapid user interaction prototyping mechanism for capturing requirement from user/client?

Section (2)- Prototype Tool Information.

Please rate your agreement with the following statements about how you feel in general when using ***Marama AI:Automated inconsistency Checker*** (a new automated tool to manage requirements and check inconsistency). Please circle or tick the level of agreement that applies using the following scale:

1: Strongly Disagree (SD) **2**:Disagree (D) **3**: Undecided (U) **4**: Agree (A) **5**: Strongly Agree(SA)

M. Usefulness:

It helps me be more effective in capturing requirements.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It makes it easier for me to capture requirements.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It makes it easier for me to see the outcome of a requirement.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It makes it easier for me to verify requirements with a client from an early stage.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It makes it easier for me to confirm requirements with a client from an early stage.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

N. Ease of Use:
It is easy to use.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is user friendly.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I don't notice any inconsistencies as I use the tool.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

O. Ease of Learning:
I learned to use it quickly.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I easily remember how to use it.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to learn to use it.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

P. Satisfaction:
I am satisfied with it.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I would recommend it to a friend.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is fun to use.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

Q. Cognitive Dimensions of Marama AI:

It is easy to see various parts of the tool.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to make changes.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The notation is succinct and not long-winded.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

Some things do require hard mental effort.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to make errors or mistakes.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The notation is closely related to the result.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to tell what each part is for when reading the notation.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

The dependencies are visible.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

It is easy to stop and check my work so far.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

I can work in any order I like when working with the notation.

Strongly Disagree 1--------2--------3--------4--------5 Strongly Agree

7. After completing this questionnaire, can you think of obvious ways that the Marama AI tool can be improved for its end to end rapid user interaction prototyping mechanism for capturing requirement from user/client? What are they?

Thank you for your time!

Please let us know if you have any queries about this questionnaire or the survey we are conducting. Questions or concerns can either be directed to the researcher, Massila Kamalrudin(mkam032@aucklanduni.ac.nz) or to her supervisor, Professor John Hosking (john@cs.auckland.ac.nz), Dept. of Computer Science.

REFERENCES

[1] G. Evans. Getting from use cases to code, Part 1: Use-Case Analysis. Available: http://www.ibm.com/developerworks/rational/library/5383.html,retrieved on: January 2009.

[2] S. Sendall. LBB System Use Case: check-out books. Available: http://lgl.epfl.ch/research/fondue/case-studies/lbb/uc-check-out-books.htm,retrieved on: June 2009.

[3] S.S.Pty.Enterprise Architect 8. Available: http://www.sparxsystems.com/products/ea/index.html, retrieved on: February 2011.

[4] L. L. Constantine and L. A. D. Lockwood, Software for use: a practical guide to the models and methods of usage-centered design. ACM Press/Addison-Wesley Publishing Co., 1999.

[5] G. Kotonya and I.Sommerville, Requirement Engineering Process and Techniques. West Sussex,England. John Wiley & Sons Ltd, 1998.

[6] S.S.Some, "Use Cases based Requirements Validation with Scenarios," in Proc. 13th IEEE International Conference on Requirements Engineering, IEEE Press,2005.

[7] H. Yang,A.Willis,A.D.Roeck and B.Nuseibah, "Automatic detection of nocuous coordination ambiguities in natural language requirements," in Proc. IEEE/ACM international conference on Automated software engineering, Antwerp, Belgium, ACM Press. 2010.

[8] P. S. F. Adisa, F.Sudzina, B. Johansson, "Living Requirements Space: An open access tool for enterprise resource planning systems requirements gathering," Online Information Review, vol. 34, pp. 540 - 564, 2010.

[9] A.Sampaio,R.Chityan,A.Rashid and P.rayson, "EA-Miner: a tool for automating aspect-oriented requirements identification," Proc. 20th IEEE/ACM international Conference on Automated software engineering, Long Beach, CA, USA, ACM Press. 2005.

[10] S. W. Ambler. Essential (Low Fidelity) User Interface Prototypes. Available: http://www.agilemodeling.com/artifacts/essentialUI.htm, retrieved on: June 2010.

[11] S. W. Ambler, The Object Primer: Agile Model-Driven Development with UML 2.0, 3rd ed.. New York Cambridge University Press, 2004.

[12] L. Abeti,P.Ciancarini and R.Moretti, "Wiki-based requirements management for Business Process Reengineering," in ICSE workshop of Wikis for Software Engineering 2009 (WIKIS4SE '09) 2009, pp. 14-24.

[13] C. Denger and D.M.Berry., "Higher Quality Requirements Specifications through Natural Language Patterns,"Proc. IEEE International Conference on Software-Science, Technology & Engineering, IEEE Computer Society, 2003, pp. 80.

[14] F. Fabbrini,M.Fusani,S.Genesi and G.Lami, "The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool," in Proc. Software Engineering Workshop, 2001. 26th Annual NASA Goddard, IEEE.2001, pp. 97-105.

[15] G. Engels, J. M. Küster, L. Groenewegen and R. Heckel, "“UML” 2003 - The Unified Modeling Language," in “UML” 2003 - The Unified Modeling Language, ed, 2003, pp. 356-359.

[16] R. Darimont,E.Delor,P.Massonet and A.V.Lamsweerde, "GRAIL/KAOS: an environment for goal-driven requirements engineering," Proc. 19th international conference on Software engineering, Boston, Massachusetts, United States, 1997.

[17] D. Zowgh, V.Gervasi and A.MacRae, "Using default reasoning to discover inconsistencies in natural language requirements," in Proc. Eighth Asia-Pacific Software Engineering Conference, 2001(APSEC 2001), 2001, pp. 133-140.

[18] D. Zowghi and V. Gervasi, "On the interplay between consistency, completeness, and correctness in requirements evolution," Information and Software Technology, vol. 45, pp. 993-1009. 2003.

[19] S. Liu, "Verifying Consistency and Validity of Formal Specifications by Testing," Proc. World Congress on Formal Methods in the Development of Computing Systems (FM’99). Springer-Verlag.1999,Vol.1, pp. 712-712.

[20] A. Satyajit, H.Mohanty and C.George, "Domain consistency in requirements specification," Proc.Fifth International Conference on Quality Software 2005(QSIC 2005), 2005, pp. 231-238.

[21] G. Spandiakos and A. Zisman, "Handbook of Software Engineering and Knowledge Engineering." vol.1, S. K. Chang, ed: World Publishing co, 2001, pp. 329-380.

[22] B. Nuseibeh, S.Easterbrook and A.Russo, "Leveraging Inconsistency in Software Development," Journal Computer, vol. 33, Los Alamitos,CA,USA.IEEE Press. pp. 24-29, 2000.

[23] B. Litvak,S.Tyszberowicz and A.Yehudai, "Behavioral consistency validation of UML diagrams," Proc. First International Conference on Software Engineering and Formal Methods 2003. Brisbane, Australia. 2003, pp. 118-125.

[24] A. V. Lamsweerde, R.Darimont and E.Letier., "Managing conflicts in goal-driven requirements engineering," IEEE Transactions Software Engineering, vol. 24, 1998.pp. 908-926.

[25] K. Schneider, "Generating Fast Feedback in Requirements Elicitation," in Requirements Engineering: Foundation for Software Quality, 2007, pp. 160-174.

[26] A. Egyed, "Scalable Consistency Checking Between Diagrams-The ViewIntegra Approach," Proc.16th IEEE international conference on Automated software engineering, IEEE Computer Society, 2001, p. 387.

[27] A. Kozlenkov and A. Zisman, "Are their design specifications consistent with our requirements?," Proc. IEEE International Joint Conference on Requirements Engineering 2002, 2002, pp. 145-154.

[28] A. Egyed, "Instant consistency checking for the UML," Proc. 28th international conference on Software engineering, Shanghai, China, ACM, 2006, pp. 381-390.

[29] C. Nentwich, et al., "Flexible consistency checking," ACM Trans. Softw. Eng. Methodol., vol. 12, pp. 28-63, 2003.

[30] V. Gervasi and D. Zowghi, "Reasoning about inconsistencies in natural language requirements," ACM Trans. Softw. Eng. Methodol., vol. 14, pp. 277-330, 2005.

[31] A. J. O. Markku, "Software requirements implementation and management", Software & systems engineering and their applications vol.1 à 3, , 2004 pp. 1.1-1.8 2004.

[32] G. Cysneiros and A. Zisman, "Traceability and completeness checking for agent-oriented systems," Proc. 2008 ACM symposium on Applied computing, Fortaleza, Ceara, Brazil,ACM, 2008, pp. 71-77.

[33] A. Egyed and P. Grünbacher, "Supporting software understanding with automated requirements traceability," International Journal of Software Engineering and Knowledge Engineering, vol. 15, pp. 783-810, 2005.

[34] L. G. Gnesi S, Trentanni G, Fabbrini F, Fusani M, "An automatic tool for the analysis of natural language requirements," International Journal of Computer Systems Science & Engineering, vol. 20, pp. 53-61, 2005.

[35] K. K. Breitman and J. C. S. do Prado Leite, "Ontology as a requirements engineering product," Proc. 11th IEEE International Requirements Engineering Conference 2003, 2003, pp. 309-319.

[36] K. Haruhiko and S. Motoshi, "Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach," Proc. Fifth International Conference on Quality Software, 2005.

[37] F. Meziane, et al., "Generating Natural Language specifications from UML class diagrams," Requirements Engineering, vol. 13, pp. 1-18, 2008.

[38] S. Ogata and S. Matsuura, "Evaluation of a use-case-driven requirements analysis tool employing web UI prototype generation," WSEAS Trans. Info. Sci. and App., vol. 7, pp. 273-282, 2010.

[39] J.Noble,R.Biddle, E.Tempero, "Pattern for Essential Use Cases," Victoria University of Wellington, Wellington, New zealand, April 2000.

[40] L. Susan, "Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases," Proc. Proceedings of Technology of Object-Oriented Languages and Systems - TOOLS 30,Santa Barbara,CA,USA. 1999, pp. 174-174.

[41] G. Sindre and A. L. Opdahl, "Eliciting security requirements with misuse cases," Requirements Engineering, vol. 10, pp. 34-44, 2005.

[42] A. Cockburn, "Structuring use cases with goals," Journal of Object-Oriented Programming, 1997.

[43] R. Biddle, J.Nobles and E.Tempero, "Essential use cases and responsibility in object-oriented development," Aust. Comput. Sci. Commun., vol. 24, pp. 7-16, 2002.

[44] H. Kaindl, et al., "How to Combine Requirements Engineering and Interaction Design?," Proc.16th IEEE International Requirements Engineering 2008 (RE'08). Barcelona,Catalunya,Spain, 2008, pp. 299-301.

[45] T. H. M. Geisser, N. Riegel, "Evaluating the Applicability of Requirements Engineering Tools for Distributed Software Development," Working papers of University of Mannheim for January 2007, Germany, 2007..

[46] I. Sommerville, Software Engineering, 7th Edition, International Computer Science Series, Addison Wesley, 2004.

[47] A. Finkelstein and W. Emmerich, "The future of requirements management tools," Information Systems in Public Administration and Law, 2000.

[48] X. Yufei, et al., "Research on requirement management for complex systems," in 2nd International Conference on Computer Engineering and Technology 2010 (ICCET), 2010, pp. 113-116.

[49] S. W. C.Hood, S. Fichtinger and U. Pautz, Requirements Management: The Interface Between Requirements Development and All Other Systems Engineering Processes, 1st edition.,Springer, 2007.

[50] E. S. K. Yu, "Towards modelling and reasoning support for early-phase requirements engineering," Proc. Third IEEE International Symposium on Requirements Engineering, 1997., 1997, pp. 226-235.

[51] A. Goknil, I.Kurtev and K.V.D.Berg., "A Metamodeling Approach for Reasoning about Requirements," Model Driven Architecture – Foundations and Applications, 2008, pp. 310-325.

[52] R. Darimont and A. V. Lamsweerde, "Formal refinement patterns for goal-driven requirements elaboration," Proc. 4th ACM SIGSOFT symposium on Foundations of software engineering, San Francisco, California, United States, 1996.

[53] T. C. d. Sousa, J.R.Almeida,S.Viana,J.Pav "Automatic analysis of requirements consistency with the B method," SIGSOFT Softw. Eng. Notes, vol. 35, pp. 1-4, 2010.

[54] B. Nuseibeh and S. Easterbrook, "Requirements engineering: a roadmap," Proc. Conference on The Future of Software Engineering, Limerick, Ireland, 2000.

[55] A. V. Lamsweerde, "Formal specification: a roadmap," Proc. Conference on The Future of Software Engineering, Limerick, Ireland, 2000.

[56] R. Wieringa and E. Dubois, "Integrating semi-formal and formal software specification techniques," Information Systems, vol. 23, pp. 159-178.

[57] S. Kovacevic, "UML and User Interface Modeling," in The Unified Modeling Language. «UML»'98: Beyond the Notation, 1999, pp. 514-514.

[58] M. D. Fraser, K.Kumar and V.K.Vaishnavi, "Informal and Formal Requirements Specification Languages: Bridging the Gap," IEEE Trans. Softw. Eng., vol. 17, pp. 454-466, 1991.

[59] M. G. Georgiades,A.S.Andreou,C.S.Pattichis "A requirements engineering methodology based on natural language syntax and semantics," Proc. 13th International Conference on Requirements Engineering, 2005. IEEE, 2005, pp. 473-474.

[60] B. Y. Surya, R.R.Bravoco,A.T.Chatfield and T.M.Rajkumar, "Comparison of analysis techniques for information requirement determination," Commun. ACM, vol. 31, pp. 1090-1097, 1988.

[61] M. Lang and J. Duggan, "A Tool to Support Collaborative Software Requirements Management," Requirements Engineering, vol. 6, pp. 161-172, 2001.

[62] E. Hull, et al., "DOORS: A Tool to Manage Requirements," in Requirements Engineering, Springer London, 2005, pp. 173-189.

[63] S.S.Inc. Serena. Available: http://www.serena.com/docs/repository/products/rm/wp900-001-0505.pdf, retrieved on: December 2010.

[64] B. S. Corporation. CaliberRM™ Enterprise Software Requirements Management System. Available: http://www.borland.com/us/products/caliber/index.html, retrieved on:January 2011.

[65] IBM. Rational RequisitePro A requirements management tool. Available: http://www-01.ibm.com/software/awdtools/reqpro/ , retrieved on: January 2011.

[66] S. S. J. Co. Requirements Management Tool RaQuest, Available: http://www.raquest.com/ retrieved on: January 2011.

[67] Orcanos. QPack Requirements Tool for Requirements Management. Available: http://orcanos.com/Requirements_management.htm, retrieved on: January 2011.

[68] F. Chantree, B.Nuseibah,A.de Roeck and A.Willis, "Identifying Nocuous Ambiguities in Natural Language Requirements," Proc. 14th IEEE International Requirements Engineering Conference,IEEE Press. 2006.

[69] A. T. Bahill and S. J. Henderson, "Requirements development, verification, and validation exhibited in famous failures," Syst. Eng., vol. 8, pp. 1-14, 2005.
[70] A. P. Mathur, Foundations of Software Testing: Fundamental Algorithms and Techniques: Pearson Education India, 2007.
[71] S.Robertson and A. J. Robertson, Mastering the Requirements Process, 2nd Edition, Addison-Wesley Professional, 2006.
[72] I. Sommerville, Software Engineering, 9th ed., Addison-Wesley, 2011.
[73] J. M. Atlee, S.L. Pfleeger, Software Engineering Theory and Practice Fourth Edition, Prentice Hall, 2010.
[74] A. Kozlenkov and A. Zisman, "Are their Design Specifications Consistent with our Requirements?," Proc. 10th Anniversary IEEE Joint International Conference on Requirements Engineering, IEEE Computer Society, 2002, pp. 145-156.
[75] T. Olson and J.Grundy, "Supporting Traceability and Inconsistency Management Between Software Artefacts," Proc.of the IASTED International Conference on Software Engineering and Applications, Boston, MA, November 2002.
[76] C.Nentwich, W.Emmerich and A.Finkelstein, "Consistency management with repair actions," Proc. 25th International Conference on Software Engineering, Portland, Oregon, 2003.
[77] B. Nuseibeh, S.Easterbrook and A.Russo, "Making inconsistency respectable in software development," Journal of Systems and Software, vol. 58, pp. 171-180, 2001.
[78] G.Spanoudakia and A.Zisman, "Inconsistency management in software engineering: Survey and open research issues," Handbook of software Engineering and Knowledge Engineering, World Scientific, 2001.
[79] A. D. Lucia, F.Fasano,R.Oliveto and G.tortora "Recovering traceability links in software artifact management systems using information retrieval methods," ACM Trans. Softw. Eng. Methodol., vol. 16, p. 13, 2007.
[80] M. F. Bashir and M. A. Qadir, "Traceability Techniques: A Critical Study,"Proc. Multitopic Conference, 2006(INMIC '06). IEEE, 2006, pp. 265-268.
[81] E. A. Silver, "An Overview of Heuristic Solution Methods," The Journal of the Operational Research Society, vol. 55, pp. 936-956, 2004.
[82] A .A. G. Sutcliffe and N. A. M. Maiden, "Bridging the requirements gap: policies, goals and domains," Proc. Seventh International Workshop on Software Specification and Design 1993, 1993, pp. 52-55.
[83] N. Kokash, " An introduction to heuristic algorithms, 2005, available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.105.8050, retrieved on: January 2009.
[84] C. L. Heitmeyer, R.D Jeffords and B.G Labaw, "Automated consistency checking of requirements specifications," ACM Trans. Softw. Eng. Methodol., vol. 5, pp. 231-261, 1996.
[85] M. D. Fraser,K.Kumar and V.K.Vaishnavi, "Informal and formal requirements specification languages: bridging the gap," IEEE Transactions on Software Engineering, vol. 17, pp. 454-466, 1991.
[86] W. Jirapanthong and A. Zisman, "XTraQue: traceability for product line systems," Software and Systems Modeling, September 05, 2007.
[87] A. Goknil,I.Kurtev and J.W.Veldhus, "Semantics of trace relations in requirements models for consistency checking and inferencing," Software and Systems Modeling,Vol.10,pp 31-54, 2009.
[88] G. Antoniol, G.Canfora,G.Casazza,A.D.Lucia and E.Merlo, "Recovering traceability links between code and documentation," IEEE Transactions on Software Engineering, vol. 28, pp. 970-983, 2002.
[89] Y. Koth, K.Gondow and T.Katayama, "An incremental evaluation approach to check the consistency of XML documents," Proc. IEEE International Conference on Systems, Man and Cybernetics 2002, vol. 6, 2002.
[90] R. Chitchyan,A.Rashid,P.Rayson and R.Waters, "Semantics-based composition for aspect-oriented requirements engineering," Proc. 6th international conference on Aspect-oriented software development, Vancouver, British Columbia, Canada, ACM, 2007, pp. 36-48.
[91] R. W. Waters, "MRAT: A Multidimensional Requirements Analysis Tool," MSc Dissertation, University of Lancaster, UK, 2006.

[92] P. Kroha, R.Janetzko and J.E Labra., "Ontologies in Checking for Inconsistency of Requirements Specification," Proc. Third International Conference on Advances in Semantic Processing 2009 (SEMAPRO '09), 2009, pp. 32-37.

[93] M. Sabetzadeh, S.Nejati,S.Liaskos,S>Easterbrook and M.Chechik, "Consistency Checking of Conceptual Models via Model Merging," Proc.15th IEEE International Requirements Engineering Conference 2007(RE '07), 2007, pp. 221-230.

[94] I. Groher, A.Ader and A.egyed, "Incremental Consistency Checking of Dynamic Constraints," in Fundamental Approaches to Software Engineering. vol. 6013, Springer Berlin / Heidelberg, pp. 203-217, 2010.

[95] A. Sinha, M.Kaplan, A.Paradkar and C.Williams , "Requirements Modeling and Validation Using Bi-layer Use Case Descriptions," in Model Driven Engineering Languages and Systems. vol. 5301, Springer Berlin / Heidelberg, pp. 97-112, 2008.

[96] D.Kim, "Method and Implementation for Consistency Verification of DEVS Model against User Requirement," in 10th International Conference on Advanced Communication Technology 2008 (ICACT 2008), 2008, pp. 400-404.

[97] J. Chanda,A.Kanjilal,S.Sengupta and C.Bhattacharya, "Traceability of requirements and consistency verification of UML use case, activity and Class diagram: A Formal approach," Proc. International Conference on Methods and Models in Computer Science 2009 (ICM2CS 2009), 2009, pp. 1-4.

[98] S. Jurack, L.Lambers,K.Mehner and G.Taentzer, "Sufficient Criteria for Consistent Behavior Modeling with Refined Activity Diagrams," in Model Driven Engineering Languages and Systems. vol. 5301, Springer Berlin / Heidelberg, pp. 341-355,2008.

[99] J. Whittle and J. Schumann, "Generating statechart designs from scenarios," Proc. o International Conference on Software Engineering 2000, 2000, pp. 314-323.

[100] X. Li, Z.Liu and J.He, "Consistency checking of UML requirements," Proc. 10th IEEE International Conference on Engineering of Complex Computer Systems 2005 (ICECCS 2005), 2005, pp. 411-420.

[101] C. Zapata, G.Gonzales and A.Gelbukh, " A rule-based system for assessing consistency between UML models" Proc. 6th Mexican international conference on Advances in artificial intelligence(MICAI 2007), 2007, pp. 215-224.

[102] X. Blanc, I.Mounier, A.Mougenot and T.Mens, "Detecting model inconsistency through operation-based model construction," Proc of the 30th international conference on Software engineering, Leipzig, Germany, ACM, 2008, pp. 511-520.

[103] G. Engels, R.heckel and J.Kuster, "Rule-Based Specification of Behavioral Consistency Based on the UML Meta-model," in «UML» 2001 — The Unified Modeling Language. Modeling Languages, Concepts, and Tools, ed, 2001, pp. 272-286.

[104] I. Ha and B. Kang, "Cross Checking Rules to Improve Consistency between UML Static Diagram and Dynamic Diagram," in Intelligent Data Engineering and Automated Learning – IDEAL 2008. vol. 5326, Springer Berlin / Heidelberg, 2008, pp. 436-443.

[105] K. Ryndina, J.Kuster and H.Gall, "Consistency of Business Process Models and Object Life Cycles," in Models in Software Engineering, 2007, pp. 80-90.

[106] M. El-Attar and J. Miller, "Producing robust use case diagrams via reverse engineering of use case descriptions," Software and Systems Modeling, vol. 7, pp. 67-83, 2008.

[107] G. Perrouin,B.Baudry,E.Brottier and Y.Le Traon , "Composing Models for Detecting Inconsistencies: A Requirements Engineering Perspective," in Requirements Engineering: Foundation for Software Quality, 2009, pp. 89-103.

[108] K. Mehner, M.Monga and G.Taentzer, "Interaction Analysis in Aspect-Oriented Models," Proc.14th IEEE International Conference Requirements Engineering 2006 (RE'06), 2006. pp. 69-78.

[109] T. S. E. M. Islam A.M.El-Maddah, "Tracing Aspects in Goal Driven Requirements of Process Control Systems," Proc. 3rd International Conference on AOSD, 2004.

[110] J. Grundy, "Aspect-oriented requirements engineering for component-based software systems," Proc. IEEE International Symposium on Requirements Engineering 1999, 1999, pp. 84-91.

[111] Y. Yu, "From Goals to Aspects: Discovering Aspects from Requirements Goal Models," Proc. 12th IEEE International Requirement Engineering Conference 2004, 2004.pp. 38-47.

[112] C. Zerong and A. Ghose, "Web agents for requirements consistency management," Proc. IEEE/WIC International Conference on Web Intelligence 2003 (WI 2003), 2003, pp. 710-713.

[113] F. Taibi,K.D.Jacob and A.F.Mohammed, "On checking the consistency of Object-Z classes," SIGSOFT Softw. Eng. Notes, vol. 32, p. 11, 2007.

[114] K. Kaneiwa and K. Satoh, "On the complexities of consistency checking for restricted UML class diagrams," Theoretical Computer Science, vol. 411, pp. 301-323, 2010.

[115] K. Mu, W.Liu,Z.Jin,R.Lu,A.Yu and D.Bell,"A Merging-Based Approach to Handling Inconsistency in Locally Prioritized Software Requirements," in Knowledge Science, Engineering and Management, 2007, pp. 103-114.

[116] F. Weitl, M.Jaksic and B.Freitag , "Towards the automated verification of semi-structured documents," Data & Knowledge Engineering, vol. In Press, 2008.

[117] J. Scheffczyk, U.M.Borghoff,A.Birf and J.Siedersleben, "Pragmatic consistency management in industrial requirements specifications," Proc. Third IEEE International Conference on Software Engineering and Formal Methods 2005 (SEFM 2005), 2005, pp. 272-281.

[118] T. Mens and P. Van Gorp, "A Taxonomy of Model Transformation and its Application to Graph Transformation," Electronic Notes in Theoretical Computer Science, vol. 152, pp. 125-142, 2006.

[119] X. Chen, "Extraction and visualization of traceability relationships between documents and source code," Proc. IEEE/ACM international conference on Automated software engineering (ASE'10), Antwerp, Belgium, ACM.2010.

[120] J. Grundy, J.Hosking, J.Huh, and N. Li, "Marama: an Eclipse meta-toolset for generating multi-view environments," in 2008 IEEE/ACM International Conference on Software Engineering, Liepzig, Germany, May 2008.

[121] D. W. Brown, An Introduction to Object- Oriented Analysis object and UML in Plain English, second ed. New York: John Wley& Sons,Inc, 2002.

[122] L. L. Constantine, "Essential modeling: use cases for user interfaces," interactions, vol. 2, pp. 34-46, 1995.

[123] L. L. Constantine and A. D. L. Lockwood, "Structure and style in use cases for user interface design," in Object modeling and user interface design: designing interactive systems, Addison-Wesley, Longman Publishing Co., Inc., 2001, pp. 245-279.

[124] S. Aithal. S. Vinay, P. Desai, "An Approach towards Automation of Requirements Analysis," Proc. International MultiConference of Engineers and Computer Scientists, Hong Kong, 2009, pp. 1080-1085.

[125] S. S. Some, "Use cases based requirements validation with scenarios," Proc. 13th IEEE International Conference in Requirements Engineering 2005, 2005, pp. 465-466.

[126] R. C. Bjork. (1998, June). Use Cases for Example ATM System. Available: http://www.math-cs.gordon.edu/courses/cs320/ATM_Example/UseCases.html, retrieved on: February 2009.

[127] M. Glinz, "A lightweight approach to consistency of scenarios and class models," 2000. Proc.4th International Conference on Requirements Engineering 2000, 2000, pp. 49-58.

[128] T.Horton. Example Use Cases for PARTS. Available: http://www.cs.virginia.edu/~horton/cs494/examples/parts/usecases-ex1.html, retrieved on: February 2009.

[129] J. Kim, S.Park and V.sugumaran, "Improving use case driven analysis using goal and scenario authoring: A linguistics-based approach," Data & Knowledge Engineering, vol. 58, pp. 21-46, 2006.

[130] Scenario examples. Available: http://www.opensrs.com/resources/documentation/sync/scenarioexamples.htm, retrieved on: February 2009.

[131] W. L. Poon and A. Finkelstein, "Consistency management for multiple perspective software development,"Proc. Joint proceedings of the second international software architecture workshop (ISAW-2) and international workshop on multiple perspectives in software development (Viewpoints '96) on SIGSOFT '96 workshops, San Francisco, California, United States, ACM, 1996, pp. 192-196.

[132] A. Finkelstein, "A Foolish Consistency: Technical Challenges in Consistency Management," in Database and Expert Systems Applications, 2000, pp. 1-5.

[133] J. Grundy, J.Hosking and W. B. Mugridge, "Inconsistency management for multiple-view software development environments," IEEE Transactions on Software Engineering, vol. 24, pp. 960-981, 1998.

[134] S. F. Tjong, N.Hallam and M.Hartley, "Improving the Quality of Natural Language Requirements Specifications through Natural Language Requirements Patterns," Proc.

The Sixth IEEE International Conference on Computer and Information Technology, 2006 (CIT '06), 2006, pp. 199-199.

[135] B. Nuseibeh, S.Easterbrook and A.Russo, "Making inconsistency respectable in software development," Journal of Systems and Software, vol. 58, pp. 171-180, 2001.

[136] Z. Huzar, L.Kuzniarz, G.Reggio and J.L.Sourrouille "Consistency Problems in UML-Based Software Development," in UML Modeling Languages and Applications, ed, 2005, pp. 1-12.

[137] A.Mehra,J.Grundy and J.Hosking, "A generic approach to supporting diagram differencing and merging for collaborative design," Proc. 20th IEEE/ACM international Conference on Automated software engineering, Long Beach, CA, USA, 2005.

[138] A. Ghose, G.Koliadis and A.Chueng, "Process Discovery from Model and Text Artefacts," Proc. IEEE Congress on Services 2007, pp. 167-174.

[139] A. Kozlenkov and A. Zisman, "Are their design specifications consistent with our requirements?," proc. IEEE Joint International Conference on Requirements Engineering, 2002, 2002, pp. 145-154.

[140] G. Engels, R. Heckel, and J.M.Kuster, "The Consistency Workbench: A tool for consistency management in UML-based development," in UML 2003—the Unified Modeling Language. vol. 2863, New York, Springer, Berlin Heidelberg 2003, pp. 356–359.

[141] A. Katasonov and M. Sakkinen, "Requirements quality control: a unifying framework," Requirements Engineering, vol. 11, pp. 42-57, 2006.

[142] S. Ogata and S. Matsuura, "Evaluation of a use-case-driven requirements analysis tool employing web UI prototype generation," WSEAS Transactions on Information Science and Applications, vol. 7, pp. 273-282, 2010.

[143] Z. Jia, C.K.Chang and J.Y.Chung, "Mockup-driven fast-prototyping methodology for Web requirements engineering," Proc. 27th Annual International Computer Software and Applications Conference 2003 (COMPSAC 2003), 2003, pp. 263-268.

[144] S. Xiping, "S-RaP: A Concurrent Prototyping Process for Refining Workflow-Oriented Requirements," Proc. 13th IEEE International Conference on Requirement Engineering 2005, 2005, pp. 416-420.

[145] N. Sukaviriya, et al., "User-Centered Design and Business Process Modeling: Cross Road in Rapid Prototyping Tools," in Human-Computer Interaction – INTERACT 2007. vol. 4662, Springer Berlin / Heidelberg, 2007, pp. 165-178.

[146] R. V.Buskirk and B. W. Moroney, "Extending prototyping," IBM Systems Journal, vol. 42, pp. 613-623, 2003.

[147] L.L Constantine, "Rapid Abstract Prototyping," http://www.foruse.com/articles/abstractprototypes.pdf, retrieved on: February 2009

[148] C. Boghdan, "Generating an Abstract User Interface from a Discourse Model Inspired by Human Communication," Proc. 41st Annual Hawaii International Conference on System Sciences 2008(HICSS'08), IEEE Computer Society,2008,pp. 36-36

[149] T. Memmel and H. Reiterer, "Inspector: Interactive UI Specification Tool," in Computer-Aided Design of User Interfaces VI, 2009, pp. 163-175.

[150] L.L.Constantine and A. D. L. Lockwood, "Usage-centered software engineering: an agile approach to integrating users, user interfaces, and usability into software engineering practice," Proc. 25th International Conference on Software Engineering, Portland, Oregon, 2003.

[151] G. Gabrysiak, H.Giese and A.Seibel,"Interactive Visualization for Elicitation and Validationn of Requirements with Scenario-Based Prototyping," Proc. 2009 Fourth International Workshop on, Requirements Engineering Visualization (REV), 2009, pp. 41-45.

[152] D. Li, X.Li,J.liu and Z.liu, "Validation of requirement models by automatic prototyping," Innovations in Systems and Software Engineering, vol. 4, pp. 241-248, 2008.

[153] J. Vijayan and G. Raju, "Requirements Elicitation Using Paper Prototype," in Advances in Software Engineering. vol. 117, Springer Berlin Heidelberg, 2010, pp. 30-37.

[154] S. Melia, O. Pastor. P. J. Molina, "Just-UI: A User Interface Specification Model," Proc. Computer Aided Design of User Interfaces III (CADUI'2002), Valenciennes, France,pp.63-74, 2002.

[155] S. Dream Ltd. Creative New Media. Available: http://www.silicon-dream.com/, retrieved on: January 2011.

[156] A.Lund.(1998). USE Questionnaire Resource Page. Available: http://usesurvey.com/IntroductionToUse.html, retrieved on: February 2010.

[157]A.Blackwell,C.Britton,A.Cox,T.Green,C.Gurr,G.Kadoda,M.Kutar,M.Loomes,C.Nehaniv,M.Petre,C.Roast,C.Roe,A.Wong and R.Young, "Cognitive Dimensions of Notations: Design Tools for Cognitive Technology," in Cognitive Technology: Instruments of Mind. vol. 2117, Springer Berlin / Heidelberg, 2001, pp. 325-341.

[158] T. Green, A. Blackwell., Cognitive Dimensions of Information Artefacts:a tutorial. Version 1.2,1998.Available: https://www.cl.cam.ac.uk/~afb21/CognitiveDimensions/CDtutorial.pdf, retrieved on:February 2010.

[159] J. Ramey, T.Boren,E.Chuddihy,J.Dumas,Z.Guan,M.J.V.D.Haak and M.D.T.D.Jong, "Does think aloud work?: how do we know?," extended abstracts Human Factors in Computing Systems (CHI EA'06), Canada, ACM, 2006.

[160] M. Kutar, C. Britton., J. Wilson, "Cognitive Dimensions An Experience Report," in Twelfth Annual Meeting of the Psychology of Programming Interest Group, Memoria, Cozenza Italy, 2000, pp. 81-98.

[161] N. M. Ali,J.Hosking,J.Grundy and J.Huh, "End-user oriented critic specification for domain-specific visual language tools," Proc. IEEE/ACM international conference on Automated software engineering, Antwerp, Belgium, 2010.

[162] INCOSE International Council on system Engineering, INCOSE Requirements Management Tools Survey, Available: http: http://incose.org/ProductsPubs/products/rmsurvey.aspx

[163] P.A.Laplante, Requirements Engineering for Software and System, CRC Press,2009.

[164] N.M.Ali, “A Generic Visual Critic Authoring Tool” Proc. IEEE Symposium on Visual Languages and Human-Centric Computing (VLHCC '07),USA, 2007.

[165] R.L.Baskerville, “Investigating information systems with action research” Journal Communications of the AIS, Volume 2 Issue 3es,Nov.1999

[166] D.E. Avison,F.Lau,M.D.Myers and P.A.Nielsen,”Action Reserach”, Magazine Communication of the ACM, Volume 42 Issue 1,Jan.1999.

[167] Institute of Electrical and Electronics Engineers, “IEEE guide to software Requirements Specification”Standard 830-1984,New York, IEEE Computer Society Press,1984.

[168] A. Davis, S. Overmyer, K. Jordan, J. Caruso, F. Dandashi, A. Dinh, G.Kincaid, G. Ledeboer, P.Reynolds,.P.Sitaram, A. Ta, and M.Theofanos,”Idnetifying and Measuring Quality in a Software Requirements Specifications, Proc. International Software Metrics Symposium,Pages 141-152,Los Almitos,CA,USA,1993.IEEE Computer Society.