Tool Support for Essential Use Cases to Better Capture Software Requirements

Massila Kamalrudin
Department of Electrical and Computer Engineering,
University of Auckland
Private Bag 92019, Auckland,
New Zealand
(64 9)373-7599 ext 89427
mkam032@aucklanduni.ac.nz

John Grundy
Centre for Complex Software Systems and Services,
Swinburne University of Technology
PO Box 218, Hawthorn, Victoria 3122, Australia
+61 3 9214 8731
jgrundy@swin.edu.au

John Hosking
Department of Computer Science,
University of Auckland
Private Bag 92019, Auckland,
New Zealand
(64 9)373-7599 ext 88297
john@cs.auckland.ac.nz

ABSTRACT
Capturing software requirements from clients often leads to error prone and vague requirements documents. To surmount this issue, requirements engineers often choose to use UML models to capture their requirements. In this paper we discuss the use of Essential Use Cases (EUCs) as an alternative, user-centric representation which was developed to ease the process of capturing and describing requirements. However, EUCs are not commonly used in practice because, to our knowledge, no suitable tool support has been developed. In addition, requirements engineers face difficulties in finding the correct "essential" requirements (abstract interactions) in a time efficient manner. In order to overcome these problems, we have developed a prototype tool for automated tracing of abstract interactions. We describe the tool and compare the performance and correctness of the results provided by it to that of manual essential use case extraction efforts by a group of requirements engineers. The results of an end user study of the tool's usefulness and ease of use are also discussed.

Categories and Subject Descriptors
D.2.1 Requirements/Specifications; D.2.2 Design Tools and Techniques; D.2.6 Programming Environments

General Terms
Design; Human Factors

Keywords
Requirements Extraction, Essential Use Cases, Automated Tracing Tool.

ASE'10, September 20–24, 2010, Antwerp, Belgium.

1. INTRODUCTION
When capturing software requirements from clients, requirements engineers often use some form of natural language, written either by clients or themselves. These form a human-centric representation of the requirements accessible to both 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. Engineers thus often use models to represent these informally expressed requirements which allow for better checking, analysis and structured representations, ideally leading to higher quality systems engineered from them.

There are many ways of representing 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" [1] 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" [1]. UML models are a common way of capturing software requirements [2] 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 [2]. Various studies have determined that eliciting requirements and extracting their use cases can be arduous and can lead to rather imprecise analysis [3],[4],[5],[6]. Due to these deficiencies, Constantine and Lockwood [2] were motivated to develop the Essential Use Case (EUC) modeling approach to overcome some of these problems. Although the usage of EUCs is not as widespread as are conventional use cases, several researchers have recommended their adoption as their use helps in integrating the requirement engineering and interaction design processes [3],[7],[8]. 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 modeling approaches [3],[7].

This motivated us to (1) conduct a user study to gauge requirements engineers' ability to use the EUC modeling approach to extract structured requirements from natural language; (2) develop a tool to support them to do EUC modeling

and (3) evaluate the tool to demonstrate that it enhances their ability to us EUCs effectively. In this paper, we begin by describing the initial user study with requirements engineers using the EUC approach to extract requirements and comment on some surprising findings. Then we present our prototype extraction tool and describe an experiment comparing its performance in extracting EUC models from the same requirements sample. We discuss implications of these studies and prototype and discuss our intended future work extending the tool.

2. BACKGROUND

2.1 Essential Use Cases

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*" [9]. 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 only allowing specific detail that is relevant to the intended design to be captured [7] . 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 organized into an interaction sequence. Consequently, an EUC specifies the sequence of the abstract steps and captures the core part of the requirements [7]. 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" [7]. 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 [9]. It also allows more rapid development to happen as by using EUC it is not necessary to design an actual user interface [7].

Figure 1 shows an example natural language requirement (left hand side) and an example Essential Use Case (right hand side) capturing this requirement (adapted from [10]). On the left is the natural language requirement from which important phrases are extracted (highlighted). From each of these, a specific key phrase (essential requirement) is abstracted and is shown in the Essential Use case on the right as user intentions and system responsibilities. These abstract away from specific technologies, such as typing in login information, to a more abstract expression of each requirement, such as "identify self". This opens up the possibility of alternative designs, such as using biometrics as an identification method, that still meet the "essential" requirements.

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" [3]. Requirements engineers need to abstract the essential requirements (using the EUC concept of *abstract interactions*) manually. This means dealing with understanding the natural language requirements and then extracting an appropriately abstract essential requirement embedded in an appropriate interaction sequence. To understand better the difficulty in achieving this, we have conducted a user study of postgraduate students experienced in requirements elicitation and observed both their correctness and time duration in undertaking Essential Use Case analyses manually.

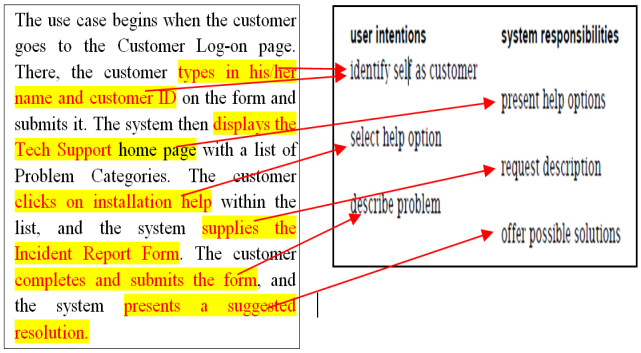

Figure 1. (Left) Example natural language requirements and (right) example Essential Use Cases.

2.2 Applying Essential Use Cases: A Study

Previous research on the EUC approach and practice in their use to model software requirements have indicated that requirements engineers have some challenges in identifying the "abstract interactions" used by EUCs and their sequencing [3]. This observation, while intuitive, is anecdotal, so to obtain a more rigorous understanding of these difficulties, we decided to conduct 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 performance and experiences in using EUCs. We used the same requirements as described in [10] 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 industry as developers and/or requirements engineers. All were familiar with UML use case modeling and most had used UML use cases to model requirements previously. None were familiar with the EUC modeling approach. Each participant was given a brief tutorial on the EUC approach and some examples of natural language requirements and derived EUC models. The participants were asked to develop an EUC model from the Natural Language requirements and we tracked their time taken and analyzed the correctness of their resulting models. The particular scenario we gave them to analyze in this evaluation was Constantine and Lockwood's "getting cash" scenario. 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.

Table 1 summarizes the results of our study. The correctness (Y for correct, x for incorrect) and time taken was recorded for each person. A correct answer (Y) means that the answer provided by the participant is same or very similar to the interaction pattern provided by a library pattern that we obtained from Constantine and Lockwood [9]. Summarizing these results:

1. The number of correct interactions 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 variability in the time taken.

Table 1. EUC extraction study results

Candidate	Answers												Time taken (minutes)
	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
	5	**6**	**4**	**7**	**4**	**7**	**6**	**5**	**9**	**2**	**3**	**8**	**123**

Average time:123/11=11.2

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 except one of the participants 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) managed to obtain a solution 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). The study also demonstrates that it is quite time consuming for participants as they need to figure out appropriate keywords that describe each abstract interaction and to organize these into an appropriate sequence of user intentions and system responsibilities. We can see that there is considerable variation in time taken and the longest time taken also does not ensure the correctness of the answer. For example the participant who took the longest (25 minutes) to accomplish the task only provided 1 correct essential interaction characterization out of 6, a poor result, while one of the better participants took only 5 minutes to produce 4 out 6 correct interactions. Our survey thus supports the anecdotal findings reported in [3] with more quantitative evidence.

3. OUR APPROACH

We were quite surprised by the results in section 2. Many of the participants were experienced industry and academic requirements modelers and all were familiar with and most experienced with using UML use case modeling. Given this background, we expected much more accurate modeling 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 [3]. This has provided us with the motivation to develop an approach and supporting tool with which to enable requirements engineers to extract accurate EUC abstract interactions automatically from textual natural language requirements.

Key research questions we had included:

- Existing Natural Language Processing (NLP)-based tools to convert textual requirements into models are often limited in terms of interactivity, the structure of the derivative models produced and the quality of the extracted models. Can a lighter-weight extraction process be adopted that considers the target EUC-based requirements representation model and/or requirements domain to provide an accurate abstraction of text requirements to essential interactions?
- Can this extraction technique be embodied in a tool that allows requirements engineers to extract essential interactions from textual requirements quickly and accurately, refine the extracted interactions, visualize the interactions as EUC models, and make changes to the models and/or textual requirements, keeping these consistent?
- Do target users evaluating this EUC requirements extraction tool find it assists the extraction of EUC models and the improvement in quality of both the extracted models and the textual natural language requirements?

In determining our approach to this, we decided NOT to use a heavy-weight Natural Language Processing (NLP) tool and formal method technique to do this extraction. This is based on a number of studies showing both the difficulty in doing this is and that the results of such an approach are often imprecise and inconsistent [11],[12],[13]. While our approach described here does not itself employ heavy-weight NLP it also does not preclude using such techniques to augment our tool in future.

Instead of using conventional NLP-based approaches we adopted a more domain specific approach. Extracting EUC essential interactions from natural language text constrains the problem domain to a set of suitable interaction descriptions. This meant we chose to develop a library of "proven" essential interactions expressed as textual phrases, phrase variants and limited regular expressions. This library of *abstract interaction patterns* was developed from a collection of such patterns previously identified by Constantine and Lockwood [9], Biddle et.al. [3] and also patterns that were developed by us and which together are applicable across various domains.

Each essential interaction pattern in the library was also associated with a collection of alternative sequences of textual requirement phrases that could match to the pattern. Each of these sequences relates to a more concrete version of the abstract interaction pattern. The textual natural language requirements were then analyzed by matching against the concrete versions looking for a good match. Abstraction could then be undertaken by instantiating 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 analyze the natural language requirements than NLP approaches but which is able to provide a set of meaningful abstract interactions to the requirements engineer. The abstract interaction patterns can be added to in order to improve our ability to recognize 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 sensible EUCs. This is as opposed to the results from the preliminary study reported in Section 2. where most users tend to provide wrong answers rather than right answers.

After extracting a set of candidate essential interaction phrases and assembling these into a candidate sequence of abstract interactions, the requirements engineer is presented with this list of interactions with the original textual natural requirements juxtaposed on the screen. The engineer can then select abstract interactions and see the natural language text these were derived from or vice versa. The engineer can move interactions and add or delete interactions. Limited update of the natural language text is also supported: the engineer can modify the natural language text and see the impact on the re-extracted essential interactions. An Essential Use Case visualization is also provided conforming to Constantine and Lockwood's approach. It can also be edited with limited update of the essential interactions it is derived from and consequently the natural language text phrases. Update of the natural language text results in update of the extracted essential interactions and Essential Use Case models.

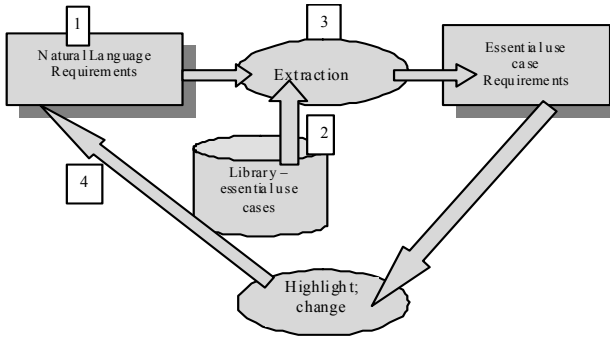

Figure 2. Our essential interaction extraction approach.

Figure 2 illustrates this extraction/trace-forward/trace-back process that we provide to requirements engineers. 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 interactions (3). The engineer can select items in textual natural language requirements of EUC interactions and see corresponding items (4). We plan to add further analysis of the EUC requirements using Essential Use Case patterns and support mapping of the EUC models to other requirements models in the next stage of our research.

4. TOOL SUPPORT

We have developed a prototype EUC essential interaction extraction tool based on the approach we outlined in the previous section. The idea is for requirements 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 phrase and interaction descriptive text. 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. Guidelines of using the tool and the patterns are also codified. Moreover, engineers need to have an understanding of the Essential Use Case concept and methodology before using the tool. 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.

4.1 Tool Process

The framework for extraction and trace-forward and trace-back between the abstract interactions from the textual natural language requirements and vice versa is illustrated in Figure 3. We use the scenario of getting cash by [10] 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 within 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 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 an instance of the abstract interaction into the EUC model, to the right in Figure 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 supports both traceability between textual natural language and EUC model elements but also assist engineers and clients in reasoning about 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.

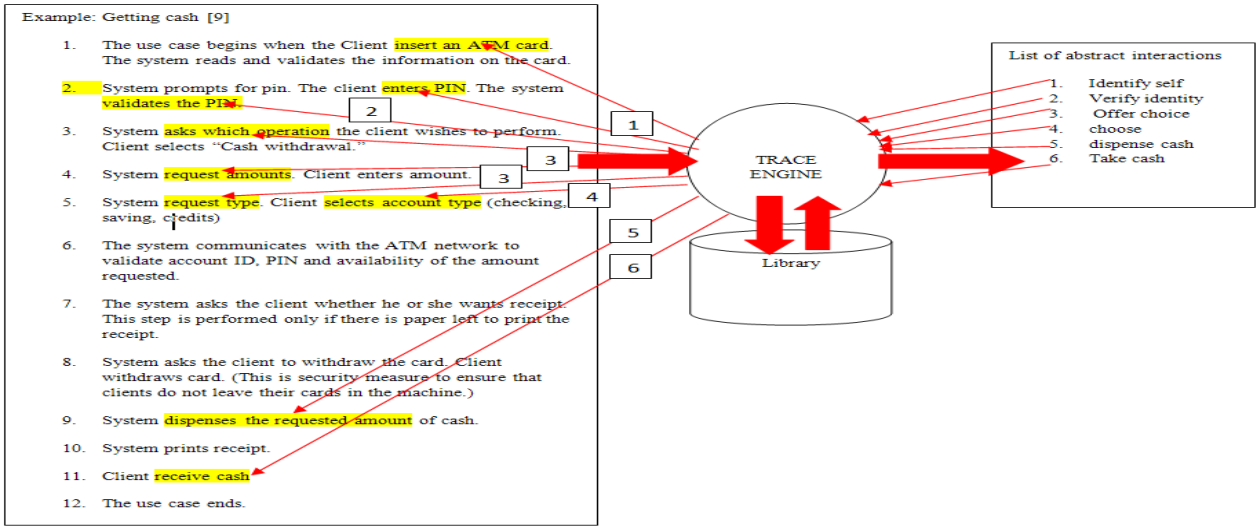

Figure 3. An example of performing an essential interaction extraction to a EUC model and supporting trace-forward/trace-back

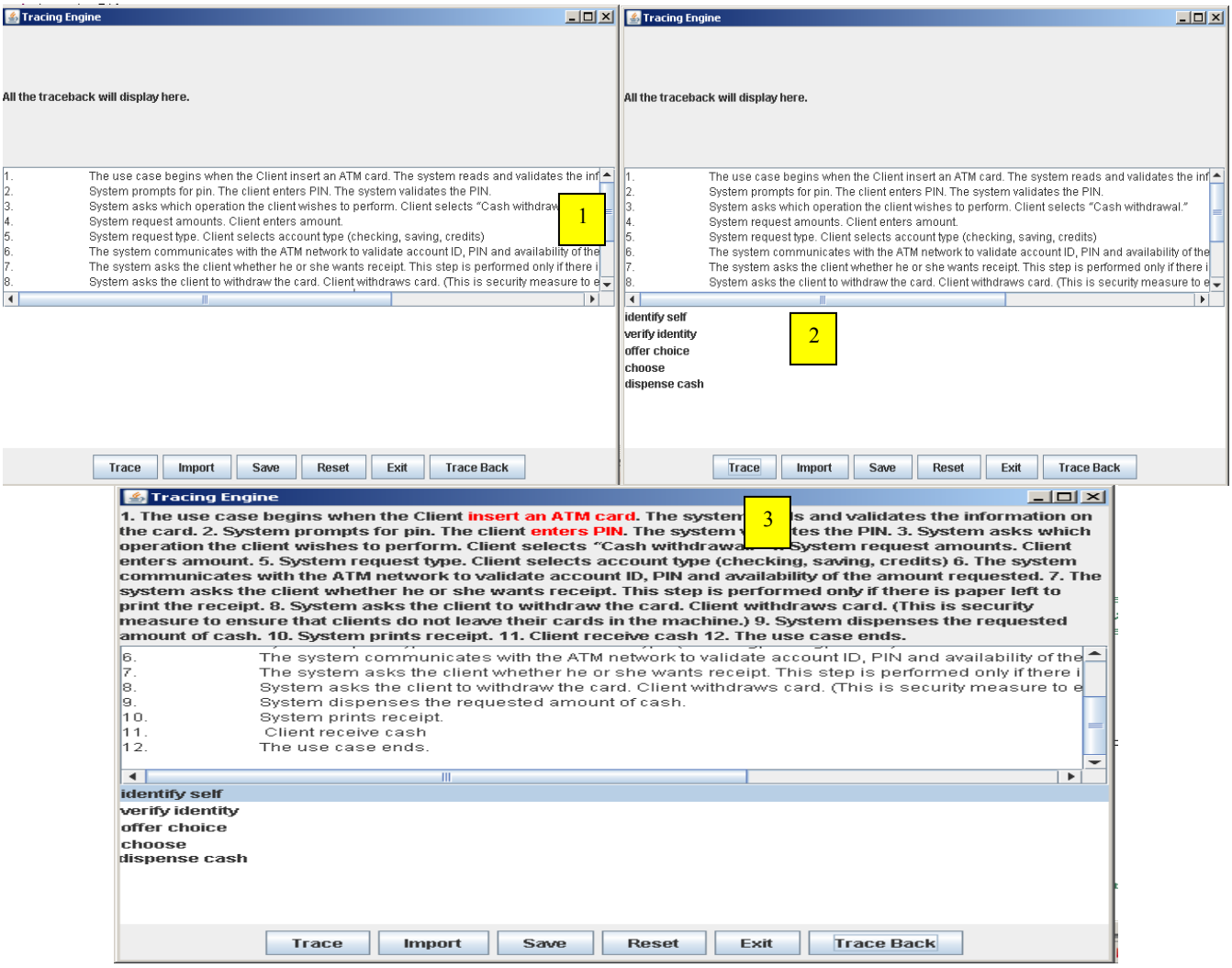

Figure 4. Our Automated Tracing Tool

4.2 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. Textual natural language requirements are written in the textual authoring tool (1). A list of corresponding essential requirements (abstract interactions) is generated automatically as shown in (2). Users can trace back each abstract interaction to the corresponding textual requirements phrases as shown in (3).

The textual natural language requirements (1) are expressed in natural language phrases. These may be including headings, numbered items, bullet points as well as sentences. In this example for clarity we use a numbered list of sentences but in general the textual natural language requirements can contain other layout as appropriate. The requirements engineer authors this textual natural language requirement either in our authoring tool, an external word processor or extracts the text from an existing document e.g. PDF, Word, and Power Point files.

The engineer then asks the tool to extract all recognized EUC "essential interactions" expressed in the textual requirements, using an essential interaction pattern database. The extracted essential interactions are shown in sequence as recognized in the text (2). Depending on the complexity of the submitted requirements text, several EUC interaction sequences, or Essential Use Cases, may be recognized. These can be divided up 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 interaction/system response divisions using Constantine & Lockwood's approach if desired.

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 users to be able 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 requirements have inconsistencies or undue repetition.

We implemented our extraction and tracing tool in Java. We have recently integrated this into a further prototype in Eclipse, Marama Essential, which provides integrated textual requirements visualization along with graphical essential interaction and EUC visualization. An example of using this tool to trace between textual requirements and extracted essential interaction and EUC model elements is shown in Figure 5. This prototype was built using our Marama meta-tools platform [28].

4.3 Essential Interactions Extraction

In order to facilitate this extraction we have developed an interaction pattern library for storing all the essential interactions and abstract interactions. We collected and categorized 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 300 phrases from various requirements 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 categorization of the phrases are on-going. Based on these 300 phrases, we have come up with close to 80 patterns of abstract interaction. On average, there are 3-4 phrases or essential interactions associated with each abstract interaction. However, some refinement on the patterns has been done and we have currently approximately 88 patterns. The full examples of patterns are documented and we may place this online at a later date.

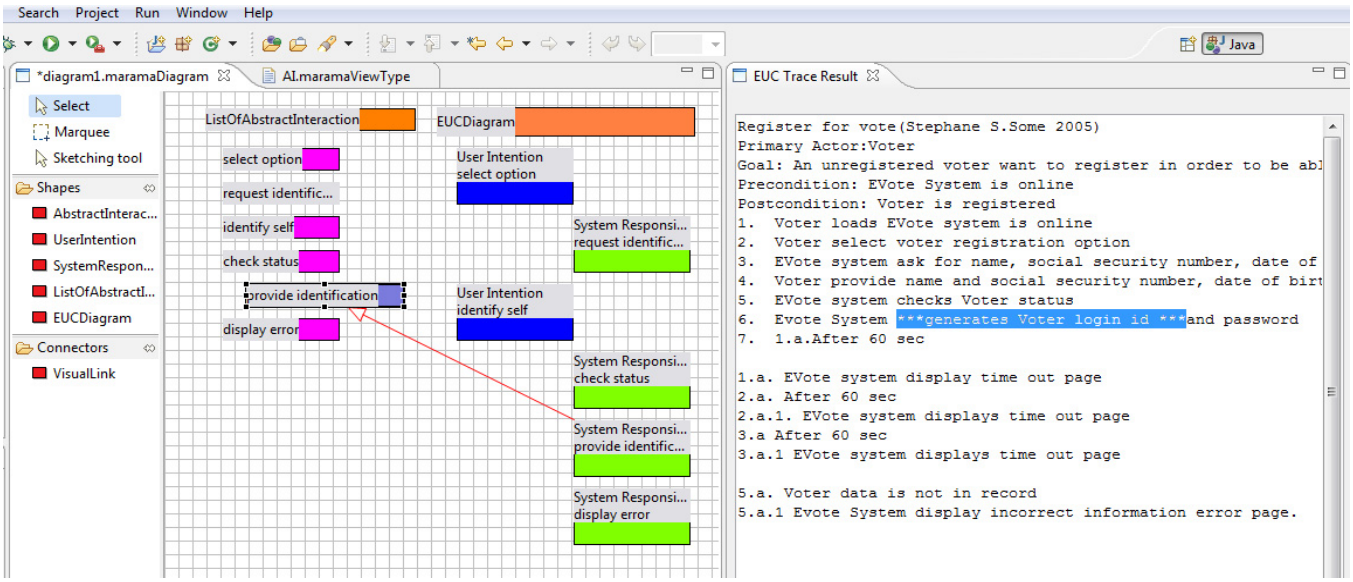

Figure 5. Example of integration into a prototype Eclipse-based EUC tracing and visualisation tool, MaramaEssential.

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 categorized based on one scenario. They have associations with five different concrete scenarios such as online business, e-commerce, online booking, online banking and online voting system. This example shows that one particular abstract interaction can be associated with multiple concrete scenarios. On average in our interaction pattern library 3-4 concrete scenarios are associated with one abstract interaction.

In order to store the essential interactions in the interaction pattern library, selected phrases ("key textual structures") are extracted from the natural language text based on their sentence structure. The 'key textual structure" uses Verb-Phrases (VP) and Noun-Phrases (NP) in the sentence structures to categorize the essential interactions. Any phrases that follow this structure will be acceptable as an essential interaction pattern in the interaction library. The tree structure of the key textual structure is illustrated in Figure 6.

The tree structure in Figure 6 shows that our library comprises three different sentence structures based on the location of the Verb Phrase (VP) and Noun Phrase (NP). The Noun Phrase can contains structure elements such as Articles (ART) and Adjectives (ADJ) or only Nouns (Noun).

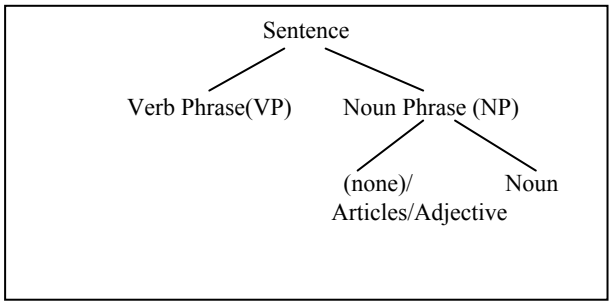

Figure 6. Tree Structure for Key Textual Phrase

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 text structure aims to provide flexibility in the library's capability to accommodate various types of sentences containing essential requirements. 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. To date we have performed this essential interaction pattern library development manually. Scope exists for semi-automating this process.

5. 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. In addition, these same participants were asked to use and evaluate the automated tracing tool using the same scenario as before. We then surveyed them to gain their perceptions of the tool's ease of use and utility for the extraction and tracing tasks evaluated.

The results in Table 2 compare the accuracy of the automated tracing tool against the previous results for manual extraction. The tool failed to identify one of the abstract interactions (Take Cash) but identified all others, providing an accuracy almost double that of the participants' average and better than all except one of the participants. The automated extraction process took just over 1 second to execute in comparison with the 11.2 minutes average taken by the manual study participants.

Results of the participant survey of the tool usefulness and ease of use are shown in Figures 7a. and 7b. 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.

Table 2. Comparison result of correctness between Manual extraction and Automated Tracing Tool

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%

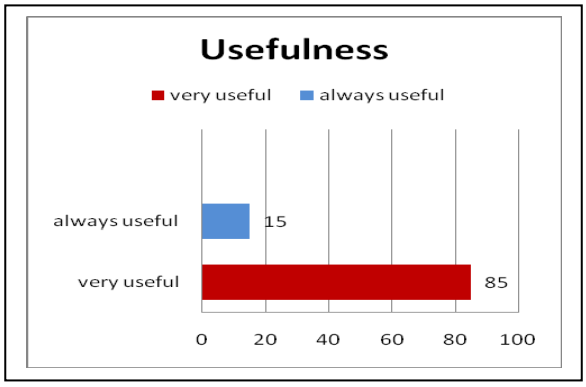

Figure 7a. The Tool Usefulness of the Automated Tracing

All participants found the automated tracing tool to be very easy (86.5%) or always easy (13.5%) to use. Qualitatively they stated that the tool was easy and simple to understand but they would have liked to have had a better user interface with a more user-friendly design rather than the preliminary prototype they used.

Most suggested that the tool could usefully be embedded within a tool that visually displays the EUCs in order to improve usability. They expected that such visualizations would allow them to better understand the interaction between the user intentions and system responsibilities in the EUC model.

The participants also evaluated the response time of the automated tracing tool for the trace back process. All found the tool to be fast or very fast, but noted some variation in speed with different scenarios when they had a chance to experiment with these.

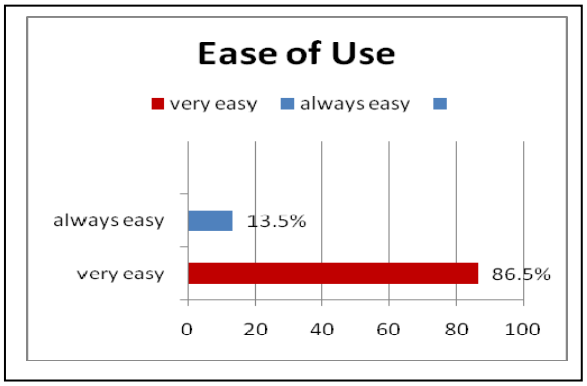

Figure 7b. The Ease of Use of the Automated Tracing Tool

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 catalog, Cellular phone [14], Voter registration [15] Cash withdrawal [16], Online book [17], Checkout book (library) [18], Seminar Enrollment [9], Transfer transaction [16], Deposit transaction [16], Assign report problem [19], Create problem report [19], Report problem [19], Booking room [20] and Place order [21]. 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 [9], Biddle et al. [3] and also with patterns that developed by us following Constantine and Lockwood's methodology. The evaluation results are shown in Figure 7c.

Figure 7c. 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. These problems, however, also lead requirement engineers to misunderstand requirements and can be one of the reasons why different requirement engineers or users will provide inconsistent results.

For example, our automated tracing tool did not derive a completely correct EUC for the scenario "Getting Cash" because of the grammar used in the sentences in the textual natural language requirements. 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 requirement 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 should have 's' but may accept without. This however complicates both the library phrase representation and authoring.

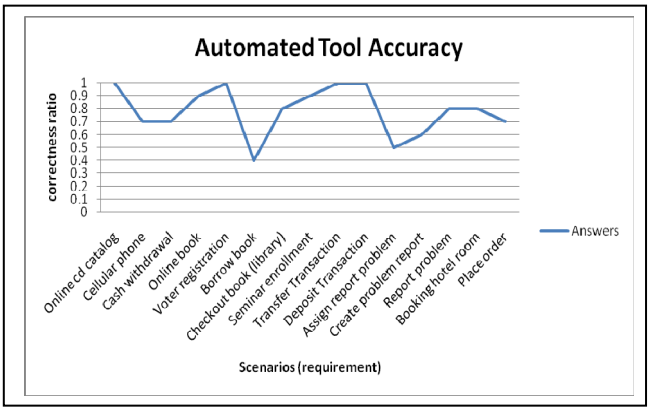

Figure 7c. Accuracy across different scenarios

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 visualized. In our Eclipse-based prototype we have experimented with adding checking for such apparent inconsistencies between requirements text and essential interactions. This is also complicated by textual requirements typically having portions of text that do not correspond directly to interactions e.g. headings, introductory or concluding remarks, comments, example input/output data, etc.

6. RELATED WORK

There have been several areas of work done in extracting UML use case models and other diagrams from Natural language requirements. Fantechi et al. [22] proposed linguistic techniques for supporting the use of semantic analysis of the use cases. Their tool succeeds in the evaluation of the natural language requirement, but does not support the problems related to the requirement consistency and completeness [22]. This work also does not deal with extracting and generating natural language requirement to the correct level of abstraction needed for an Essential Use Case model.

Harmain and Gaizaukas [23] developed a tool called CM-Builder to assist the analyst in an Object oriented environment by having a Natural language Processing technique that is able to analyze textual requirements written in English. An initial UML class diagram is extracted from the object classes that are described in the textual requirement. There are a number of limitations for this tool such as: the limited style of linguistic analysis and the limited beneficial generic knowledge used in interpreting the software requirement text; the grammar and lexicon are not adaptive, not extracting the dynamic aspect of a system, and no graphical CASE is integrated to this tool [23] . This tool is different from

our work as it is using NLP techniques in handling the natural language requirement as opposed to our lightweight approach.

Ilieva and Ormandjieva [24] proposed a methodology to process natural language requirements and map them automatically to an OO analysis model. The extracted requirement is presented in a tabular form and then transformed to the semantic network which able to be translated to an OO model (class diagram) [24]. The concept of this work is similar to our work as they are also processing natural language requirements and then transforming the requirements into a model. The differences are in terms of the technique used and the representational model that they are transforming to. We generate abstract interactions for an EUC model but they are transforming the requirement to a semantic network diagram and then a class diagram. They do not attempt to support consistency between model and the natural language.

In addition to extracting UML diagrams from natural language requirements, EUCs can also be used to capture or elicit requirements written in natural language. Using Essential Use cases (EUC) is actually not new and has been applied by some researchers in limited domains. For example, Gelperin [25] has used the Essential Use Case for analyzing and testing requirements. Gelperin agrees that EUCs effectively support user interface design [25]. He applied EUCs for checking the precision of use case. In addition, Patricio et al. [26] have used essential use cases for designing multiplatform services. They mention that the association between the identified essential use cases and the experience requirements is beneficial to understanding customer preferences. Although, EUC modeling helps in identifying areas of interaction that need to be improved together with the information which provides the concrete and objective guideline [26] for the work, all of this is done manually without any tool support.

Biddle et al.[27] believe that EUCs provide a good reusable requirements approach and they have developed a tool called UKASE, a web based use case management tool, in order to support reusability of EUCs. Guidelines and a glossary are provided for reusable use cases by exploring the requirements pattern. This tool supports the reuse of use cases, but just provides templates for users to key in the abstract interaction and does not generate the abstract interaction and EUC diagram automatically. In [3] Biddle et.al. provide a set of styles or patterns to follow when writing EUCs. One of the objectives of this paper is to help the user to write good EUCs quicker. However these researchers stated that the patterns need further development. This work also does not use any tool support and it is done by investigation and discussion based on a few scenarios. The drawbacks of this work are small and incomplete patterns for the essential and abstract interactions. We have tried to overcome the problem by developing an interaction pattern library that comprises a broader set of essential interaction patterns and abstract interaction. Our work also concentrates on developing a tool support for generating the abstract interaction automatically.

7. CONCLUSION

In this paper we have discussed the motivation for using essential use cases (EUC) to help model and structure textual natural language requirements. We have also identified some of the problems faced by requirements engineers and end users while using the EUC approach. A preliminary study of requirements engineers indicated they face problems in identifying essential interactions, sequencing these interactions, and using common name and phrase structures to describe them. They also have difficulty in tracing forwards and backwards between natural language and EUC expressed requirements.

We developed a 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 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 the collection and categorization of terminology for the library of abstract interactions. This both assists in structuring EUC expressed requirements using common terminology and also helps prevent the textual requirements from being vague and error-prone by tracing back from the EUC structured representations to the natural language text phrases.

We evaluated our prototype tool using the same group of participants as we used for the manual extraction survey. The participants evaluated the tool 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 an automated extraction and tracing tool appears to increase the ratio of correctness in extracting EUC requirements from textual natural language requirements and eases the effort of users or requirements engineers in handling the EUC, significantly reducing the time taken to develop EUC models from textual natural language requirements.

As part of our future work we are embedding our extraction approach into an integrated EUC Diagram tool (Marama Essential) developed using the Marama meta tool [28] as shown in Figure 5. This will enable users to generate and maintain the consistency of visual EUC models automatically from lists of abstract interaction. As shown in Table 2, our tool cannot guarantee correctness due to linguistic issues and common incorrectness and incompleteness of textual requirements. In order to overcome these problems, a glossary and template authoring support will also be embedded in the tool to assist improved natural language-based requirements authoring and update. We will also try considering a pre-processing phase where all different forms can be unified. We want to add additional support for inconsistency, incompleteness and redundancy detection using our extraction approach and round-trip engineering of natural language and EUC model requirements. We plan to explore a complementary approach using a composite EUC pattern template library to assist with this. As mentioned in section 2, the EUC requirements modeling approach opens up a fruitful research direction involving the consistency issue of requirements and design, encouraging engineers to check the consistency of requirements using this representation later. We plan to explore relating EUCs to further artefact views including generating UI and OO design models in our Eclipse prototype, with round-trip engineering support to consistency with textual natural language requirements.

8. ACKNOWLEDGEMENTS

This research is funded by the Ministry of Higher Education Malaysia (MOHE), Universiti Teknikal Malaysia Melaka (UTeM) the PReSS Account of the University of Auckland and the FRST Software Process and Product Improvement project.

9. REFERENCES

[1] Brown, D. W. An Introduction to Object- Oriented Analysis object and UML in Plain English. John Wiley& Sons, Inc, New York, 2002.

[2] Constantine, L. L. Essential modeling: use cases for user interfaces interactions, 2, 2 1995, 34-46.

[3] Biddle, R., Noble, J. and Tempero, E. April 2000.Pattern for Essential Use Cases. Technical Report. Victoria University of Wellington at Wellington, New Zealand.

[4] Susan, L. 1999. Use Case Pitfalls: Top 10 Problems from Real Projects Using Use Cases. In Proceedings of the Proceedings of the Technology of Object-Oriented Languages and Systems (Santa Barbara, California 1999). IEEE Computer Society, Washington, DC, USA, 465 - 466.DOI=http://doi.ieeecomputersociety.org/10.1109/TOOLS.1999.787547

[5] Sindre, G. and Opdahl, A. L. "Eliciting security requirements with misuse cases", Journal of Requirements Engineering, 2005.

[6] Cockburn, A. "Structuring use cases with goals", Journal of Object-Oriented Programming, 1997.

[7] Biddle, R., Noble, J. and Tempero, E. "Essential use cases and responsibility in object-oriented development", Australian Computer Science Communications, 2002.

[8] Kaindl, H., Constantine, L., Pastor, O., Sutcliffe, A. and Zowghi, D. 2008. How to Combine Requirements Engineering and Interaction Design? In 16th IEEE International Requirements Engineering Conference (Barcelona, Catalunya, Spain, 2008),Re'08, IEEE Computer Society, Washington, DC, USA, 299-301.DOI=http://doi.ieeecomputersociety.org/10.1109/RE.2008.59

[9] Constantine, L. L. and Lockwood, A. D. L. Software for use: a practical guide to the models and methods of usage-centered design. ACM Press/Addison-Wesley Publishing Co., 1999.

[10] Constantine, L. L. and Lockwood, A. D. L. Structure and style in use cases for user interface design. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2001.

[11] Vinay S, Aithal S, Desai. P 2009 An Approach towards Automation of Requirements Analysis. In Proceeding of International Multi Conference of Engineers and Computer Scientists.(Hong Kong, 2009) IMEC'09, IAENG, 1080-1085.

[12] Haruhiko, K. and Motoshi, S 2005. Ontology Based Requirements Analysis: Lightweight Semantic Processing Approach. In Proceedings of the Fifth International Conference on Quality Software (Melbourne, Australia, 2005). QSIC'05, IEEE Computer Society, Washington, DC, USA,223-.230. DOI=http://dx.doi.org/10.1109/QSIC.2005.46.

[13] Gnesi S, L. G., Trentanni G, Fabbrini F, Fusani M "An automatic tool for the analysis of natural language requirements." In International Journal of Computer Systems Science & Engineering, 2005.

[14] Litvak, B., Tyszberowicz, S. and Yehudai, A 2003. Behavioral consistency validation of UML diagrams. In Proceedings of First International Conference on Software Engineering and Formal Methods,(Brisbane, Australia, 2003), SEFM'03, IEEE Computer Society, Washington, DC, USA.118125,DOI=http://doi.ieeecomputersociety.org/10.1109/SEFM.2003.1236213

[15] Some.S.S.2005. Use Cases based Requirements Validation with Scenarios. In Proceedings of the 13th IEEE International Conference on Requirements Engineering, (Minnepolis/St Paul, Minnessota, USA, 2005). RE'05, IEEE Computer Society Washington, DC, USA, http://doi.ieeecomputersociety.org/10.1109/RE.2005.75

[16] Bjork R.C., Use Cases for Example ATM System,1998. http://www.math.cs.gordon.edu/courses/cs320/ATM_Example/UseCases.html

[17] Glinz, M. A lightweight approach to consistency of scenarios and class models. In Proceeding Fourth International Conference on Requirements Engineering,(Schaumburg, Illinois, June19-23 2000) (ICRE'00), IEEE Computer Society Washington, DC, USA, pp.49.DOI=http://doi.ieeecomputersociety.org/10.1109/ICRE.2000.855584

[18] Sendall. S, LBB System Use Case: check-out books, 2001. http://lgl.epfl.ch/research/fondue/case-studies/lbb/uc-check-out-books.htm

[19] Horton,T. Example Use Cases for PARTS. 2009. http://www.cs.virginia.edu/~horton/cs494/examples/parts/usecases-ex1.html

[20] Kim, J., Park, S. and Sugumaran, V. "Improving use case driven analysis using goal and scenario authoring: A linguistics-based approach." Journal of Data & Knowledge Engineering, 2006.

[21] OpenSRS,Scenarioexamples, http://www.opensrs.com/resources/documentation/sync/scenarioexamples.htm

[22] Fantechi, A., Gnesi, S., Lami, G. and Maccari, A. "Applications of linguistic techniques for use case analysis." Journal of Requirements Engineering, 2003.

[23] Harmain, H. M. and Gaizauskas, R. "CM-Builder: A Natural Language-Based CASE Tool for Object-Oriented Analysis." Journal of Automated Software Engineering, 2003.

[24] Ilieva, M. G. and Ormandjieva, O. 2005 Automatic Transition of Natural Language Software Requirements Specification into Formal Presentation. In Natural Language Processing and Information Systems, Ed Springer-Verlag, Alicante, Spain, pp. 392-397. DOI= 10.1007/b136569.

[25] Gelperin.D, Precise Use Case, 2004. www.livespecs.com.

[26] Patricio,L., Cunha, J., Fisk, R. and Pastor, O. 2003. Essential Use Cases in the Design of Multi-channel Service Offerings — A Study of Internet Banking.In Web Engineering, Ed Springer-Verlag, Oviedo, Spain, 199-206.DOI= 10.1007/3-540-45068-8.

[27] Biddle, R., Noble, J., & Tempero, E. (2002). Supporting Reusable Use Cases. In Software Reuse: Methods, Techniques, and Tools, Ed Springer-Verlag, London, UK 135-138.

[28] Grundy,J.C., Hosking, J.G. Huh, J. and Li, N. 2008.Marama: an Eclipse meta-toolset for generating multi-view environments. In Proceedings of the 30th international conference on Software Engineering, (Liepzig, Gertmany, 2008), ICSE'08, ACM Press,New York,NY,USA,819-822, http://doi.acm.org/10.1145/1368088.1368210