Improving Requirements Quality using Essential Use Case Interaction Patterns

Massila Kamalrudin
Department of Electrical and Computer Engineering, University of Auckland, Private bag 92019 Auckland 1142 New Zealand
mkam032@aucklanduni.ac.nz

John Hosking
Department of Computer Science, University of Auckland, Private bag 92019 Auckland 1142 New Zealand
john@cs.auckland.ac.nz

John Grundy
Swinburne University of Technology Centre for Complex Software, Systems & Services, PO Box 218, Hawthorn Victoria 3122, Australia
jgrundy@swin.edu.au

ABSTRACT
Requirements specifications need to be checked against the 3C's - Consistency, Completeness and Correctness – in order to achieve high quality. This is especially difficult when working with both natural language requirements and associated semi-formal modelling representations. We describe a technique and support tool that allows us to perform semi-automated checking of natural language and semi-formal requirements models, supporting both consistency management between representations but also correctness and completeness analysis. We use a concept of essential use case interaction patterns to perform the correctness and completeness analysis on the semi-formal representation. We highlight potential inconsistencies, incompleteness and incorrectness using visual differencing in our support tool. We have evaluated our approach via an end user study which focused on the tool's usefulness, ease of use, ease of learning and user satisfaction and provided data for cognitive dimensions of notations analysis of the tool.

Categories and Subject Descriptors
D.3.3 [**Software Engineering**]: Requirement/Specifications – *methodologies, tools*

General Terms
Algorithms, Documentation, Design, Human Factors, Languages, Verification.

Keywords
Requirements engineering, essential use cases, requirements patterns, consistency management, tool support

1. INTRODUCTION
Requirements specifications are captured by requirements engineers from clients at the earliest stages of software development. Requirements are most commonly in a form of natural language written by either the clients or the requirements engineers, often stored in documents, presentations and interview transcripts. This form of human-centric representation is expected to be accessible by both parties[1].

ICSE'11, May 21–28, 2011, Honolulu, Hawaii, USA.

However, a written natural language requirement is commonly error prone and vague [2] leading to *"inherent imprecision, such as ambiguities, incompleteness and inaccuracy"* [3]. It is common to be faced with inconsistencies as the requirement elicitation process involves two or more parties in delivering and understanding correct requirements[1]. Zowghi et al assert that expression by different stakeholders may lead to inconsistencies and contradictions because the parties keep changing their mind throughout the development process [4]. *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 [5], [6]. 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 [2], requirements clashes and bad dependencies [7], sometimes resulting from a lack of skills or capabilities of different users dealing with shared or related objects. These complications also often introduce *incomplete requirements* that are missing key definitions and constaints for the software system. *Incorrect requirements* may occur where 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, various quality errors and raise the cost of the system development process, often risking overall project success [6].

To address these quality problems occurring when working with natural language requirements a variety of heuristic algorithms, formal approaches and natural language processing methods have been developed and applied by industry practitioners and researchers. These deal with complex mathematical and linguistic analysis of natural language or models in order to maintain the consistency of requirements either in the form of requirements documents or models [8],[9],[10],[11],[12]. Translation of natural language models into formal or semi-formal models using these approaches allows for more rigorous inconsistency checks and completeness checks. Reflecting modified formal requirements back to stakeholders in natural language allows stakeholders to check for correctness and completeness problems, sometimes highlighting their own disagreements over the system requirements.

However, while the heavyweight techniques described above are useful, we were motivated to develop a more lightweight approach to support translation between natural language and semi-formal requirements models [13]. The previous focus of our

work was to support translation between, and then the low-level management of, consistency between three forms of requirements: textual natural language, abstract interactions, and Essential Use Case models [14]. In the research described in this paper we extend this work to better support higher level inconsistency, incompleteness and incorrectness detection and thus improve requirements quality. The essence of the approach is to support translation of natural language requirements into semi-formal abstract interaction and Essential Use Case models. We then use a concept of essential interaction patterns to detect quality problems in the extracted semi-formal requirements models (i.e. potential inconsistency, incompleteness and incorrectness), which are then highlighted to the requirements engineer and stakeholders. We highlight these potential errors by annotating the visual semi-formal model and textual natural language depictions in our support tool. We have evaluated our tool with an end user study focussing on usability and analysis using Cognitive Dimensions framework [15] heuristic characteristics.

2. BACKGROUND

We chose to use the Essential Use Cases (EUC) semi-formal model for software requirements to translate natural language requirements into; to analyze for inconsistencies, incompleteness and incorrectness; and to keep consistent with the human-centric natural language representation. EUCs are made up of a set of organized "abstract interactions" and EUCs extracted from natural language specifications can be compared against templates of "interaction patterns" to detect requirements quality problems.

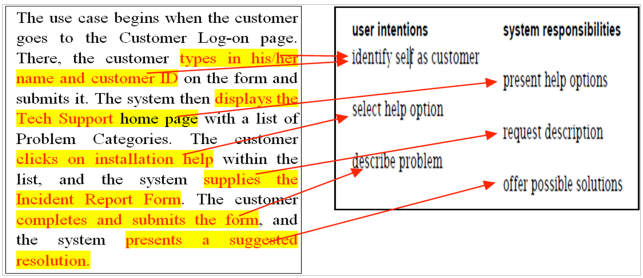

Figure 1. Capturing requirements using an Essential Use Case (adapted from [14])

2.1 Essential Use Cases

The Essential Use Case (EUC) approach was developed by Constantine and Lockwood [14]. EUCs are designed to resolve problems which occur in conventional Use Case modeling and have important benefits over that approach [16]. An EUC is defined 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*"[14]. An EUC is thus a form of dialogue between user and system which supports better communication between developers and stakeholders. Its technology-free nature enhances requirements gathering as it only allows specific detail relevant to the design to be captured [16].

Figure 1 shows an example of how an Essential Use Case is used to model a core software system requirement based on a textual natural language requirement in the form of a user scenario. Important key phrases (*essential interactions*) are extracted and identified with a meaningful abstract term (essential requirement or *abstract interaction*). The abstract interaction is then classified into a user intention or system responsibility in a sequence of abstract steps making up an EUC model of a system requirement.

An EUC is shorter and simpler compared to a conventional use case as it comprises an abstraction of only essential steps and the user's intrinsic interest. It comprises just user intentions and system responsibilities permitting users to capture the core part of the requirement without the need to describe the user interface in detail [16]. An EUC aims to identify "what the system must do" without being concerned on "how it should be done". Biddle et al. suggest there is a fruitful research area on consistency issues between the responsibility concepts in a requirement and its related design [16].

Table 1 Example of Essential interactions, Abstract interaction and their associated domains

Essential Interaction	Abstract Interaction	Scenario Domain
choose cash withdrawal	choose	Online banking
choose a type	choose	e-commerce/online banking
choose payment	choose	e-commerce/online banking/online booking
indicate the seminar	choose	online booking
indicate the vehicle type	choose	Online reservation
choose the film	choose	Online reservation
select an event	choose	Online booking/online reservation

2.2 Essential Interactions Library

As the requirements capture example of Figure 1 shows, requirements engineers need to derive appropriate *essential interactions* from the requirements at a correct level of abstraction. Biddle et al [17] and our own study [1] found that almost all users have problems defining the right level of abstraction and find the abstraction process to be time consuming. This makes checking requirements for consistency and completeness difficult.

This motivated us to provide an Essential Interaction Library to overcome these problems. This library consists of important key phrases (essential interactions) and mappings to appropriate essential requirements (abstract interactions) which support a variety of different application domains [1]. Essential interactions are not categorized based on one particular scenario but can be associated with multiple scenarios such as online booking, e-commerce, online business, online banking, online voting system, online reservation, mobile application and others. Thus, multiple essential interactions from various domains can be associated with one well-defined abstract interaction. Table 1 shows an example for the abstract interaction "choose" with multiple essential interactions and various scenario domains associated with this one interaction. Low-level requirements problems can be identified using this approach e.g. phrases of natural language text with no corresponding EUC abstract interactions identifiable or EUC interaction added by the requirements engineer with no natural language phrase(s) [1].

2.3 EUC Interaction Patterns Library

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 can be identified for a range of typical user/system interactions in a wide variety of domains [17]. 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 i.e. exemplify one common interaction style. If it deviates from this pattern this typically indicates incompleteness (missing interactions), incorrectness (wrongly sequenced interactions or wrong interactions), and possibly inconsistency (redundancy, conflicting or nonsensical interactions). Table 2 shows an example of an EUC interaction pattern for reserving items and its sequence of abstract interactions.

Table 2 Example of an EUC Interaction Pattern

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

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. Figure 2 shows an outline of our requirements quality management process.

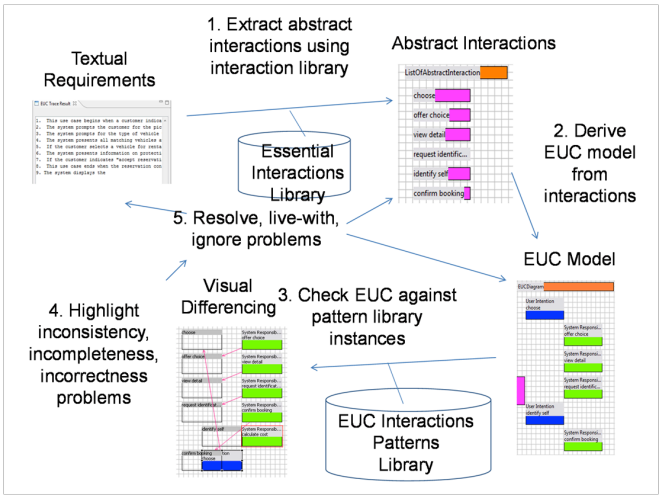

Figure 2. Outline of our requirements quality management process.

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 [13]. 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 generated. A set of inter-model checks between different requirements representation components and intra-model checks of each of the specific models 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 EUC pattern's set of abstract interactions and their sequencing. These comparisons highlight potential intra- and inter-model problems such as:

- *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 requirement. This detects inconsistencies between models where one has been edited and others not and the ordering of the interactions between user and the system needs to be made consistent (eg see Figure 4).
- *Naming of requirement elements:* The name of an EUC component must be the same as the abstract interaction or vice versa 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 EUC pattern library. This detects inconsistencies between models and also incompleteness: for example, a large textual requirement phrase with no matching abstract interaction or abstract interaction with no matching natural language phrase (eg see Figure 5).
- *EUC interaction pattern matching:* the abstract interaction elements and sequence of elements in EUC models needs 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 representation based on the matching pattern in the EUC pattern library. This detects incomplete and incorrect requirements elements: EUC models not conforming to a recognized pattern usually indicate missing, duplicated or redundant elements, or incorrectly expressed interaction components and sequences in the requirements extracted (eg see Figure 6).
- *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 e.g. EUC has start/end interactions; naming conventions of elements are met; all elements are part of a valid sequence of EUC model-compliant interactions, etc (eg see Figure 3 (2,3)).
- Consistency between 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 (eg see Figure 3 (4)).

When issues with requirements models are detected we focus on providing warning, feedback notification and visualization 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 mis-match between component and 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 are assumed to be "complete" [18] once all the requirements model elements satisfy a match or matches in the EUC interaction pattern library.

Requirements engineers can choose to either (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, or (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, changing the sequence of components of the abstract interaction or EUC, 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 correctness and completeness of the requirements model. While the EUC model is arguably end user-friendly, keeping it consistent with the natural language representation affords the latter human-centric view's continued use through the requirements engineering process.

4. TOOL SUPPORT AND USAGE

4.1 Tool Support

We have developed a prototype tool called MaramaAI (Automated Inconsistency checker) to help requirements engineers in managing inter-notation requirements translation, consistency management and quality improvement process based on our approach outlined in the previous section. Our tool helps to lessen human intervention and minimizes time taken to manage requirements formalisation from textual natural language to the semi-formal representation in an EUC model. It also supports incremental refinement of the requirements to address detected quality issues but also evolution of the requirements over time. The natural language requirements are kept consistent with the EUC model allowing both 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 understand better the flow of the interactions, structure of the requirements and 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 requirements refinement and correction.

Deeper analysis for completeness and correctness checks are provided by the tool. The tool compares extracted EUC models to our set of template EUC interaction patterns that represent valid, common ways of capturing EUC models for a wide variety of domains. Matching of 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 or else 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".

4.2 Consistency Checking

We use the example scenario of reserving a vehicle to illustrate the requirements extraction, checking and evolution process using our MaramaAI tool. The textual natural language requirements are processed and a sequence of abstract interactions extracted and visualised. We use a large database of abstract interaction patterns to identify phrases in the natural language which map onto an EUC abstract interaction concept [1]. To do this, we parse the natural language text for keyword phrases, locate a matching essential interaction phrase in the database and then use its associated abstract interaction. From the sequence of abstract interactions an EUC model is generated using the EUC model visual language.

Figure 3 shows an example of some natural language requirements (1), extracted abstract interaction phrases (2), and a generated EUC model representing the requirements (3).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 to the other two representations. Finally, it provides deeper analysis by comparing the EUC model to a library of EUC model templates and highlights deviations. We initially illustrate inter-representation consistency management and then EUC model analysis.

Figure 3 (4) shows addition of a new abstract interaction "calculate cost". A warning notifies where an inconsistency is detected between representations (5). Users can either (i) resolve the inconsistency by updating the textual natural language requirement; (ii) undo the change that introduced the inconsistency; or (iii) tolerate the inconsistency. A problem marker warning is provided to inform users about such unresolved inconsistency errors (6).

Figure 4. shows an example of MaramaAI tolerating inconsistency when EUC element sequence order is changed. The EUC element "choose" has been moved to the end of the EUC model and this change impacts the other two requirements forms. The textual requirement and abstract interaction sequence are now inconsistent with the EUC representation. The tool colors the associated abstract interaction "choose" red and annotates the associated essential interaction "indicates" with "***". An inconsistency problem marker appears notifying the user about the inconsistency. Options to resolve the inconsistency by moving the associated abstract interaction component are 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.

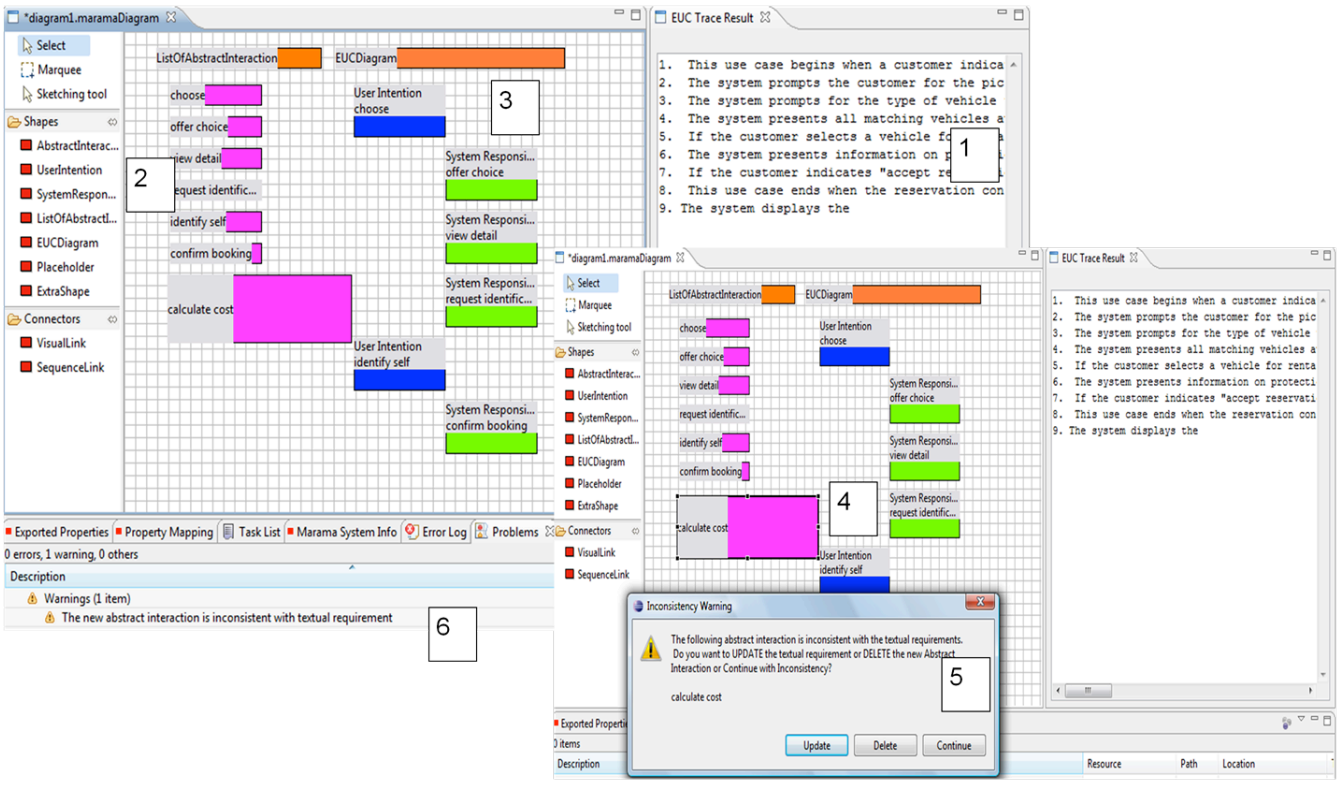

Figure 3. Extracting an EUC model then adding a new abstract interaction.

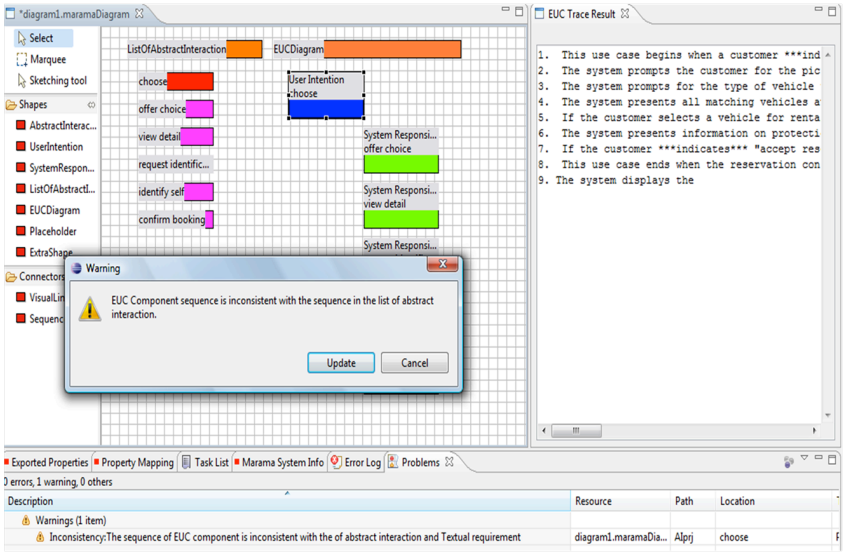

Figure 4. Changing the ordering of EUC elements.

The tool forces the user to resolve some inconsistencies if the error causes major inconsistency of the requirements components from the beginning or throughout the process. For example, in Figure 5, the abstract interaction "choose" is changed to "ask help". This change causes an inconsistency with the EUC component and textual requirement as indicated in Figure 5 (1). This is because the abstract interaction is related to a particular essential interaction in the interaction library and this may affect the extracted essential interaction. The EUC component is also affected as it is dependent on its associated extracted abstract interaction. The user is provided with a list of words or phrases (2) as alternatives for "ask help" which are recognized as correct and complete, as they match essential interaction phrases contained in the EUC interaction pattern library. MaramaAI users can create a new natural language requirement phrase based on the suggestions they think are relevant to the context provided they map to a phrase in the EUC interaction pattern library. This library can also be extended with new abstract interaction phrases as required. This is currently done by the tool developer but we will let requirements engineers do this in future via visual interfaces.

4.3 Inconsistency, Incompleteness and Incorrectness 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 our 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 [14] and Biddle et al [17] . 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 their requirements model by comparing their EUC, representing a semi-formal model of the original 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 interaction 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 possibly incorrect textual requirements.

For example, Figure 6 (1) 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 2) to compare to the extracted EUC model. Alternatively they 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 the extracted EUC requirements model is consistent with the identified EUC interaction pattern library template or not. If differences are found a warning message is provided and the tool uses a visual differencing approach [19] to highlight potential inconsistency, incompleteness and/or incorrectness errors that may exist in the requirements model, as shown in Figure 6 (2).

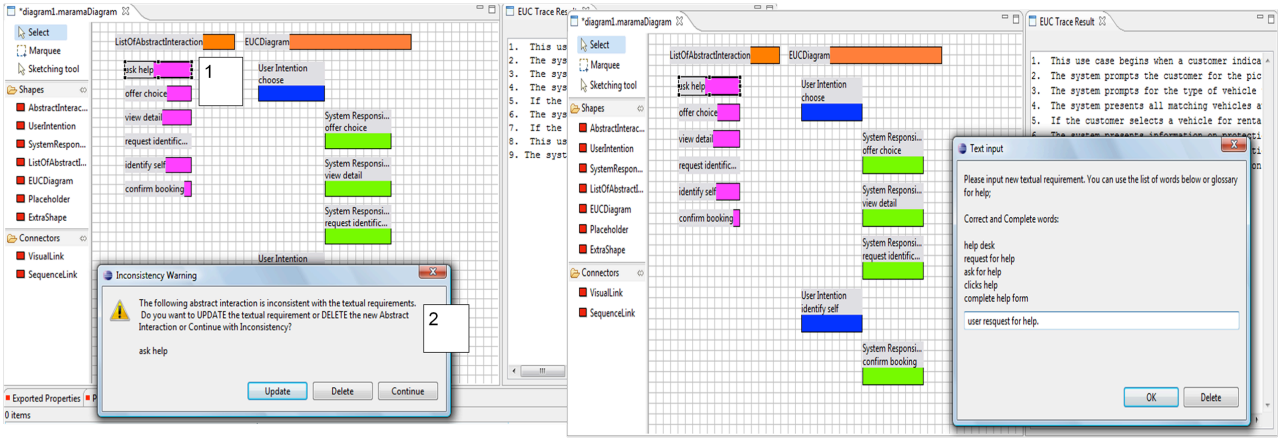

Figure 5. Change of Abstract interaction element's name

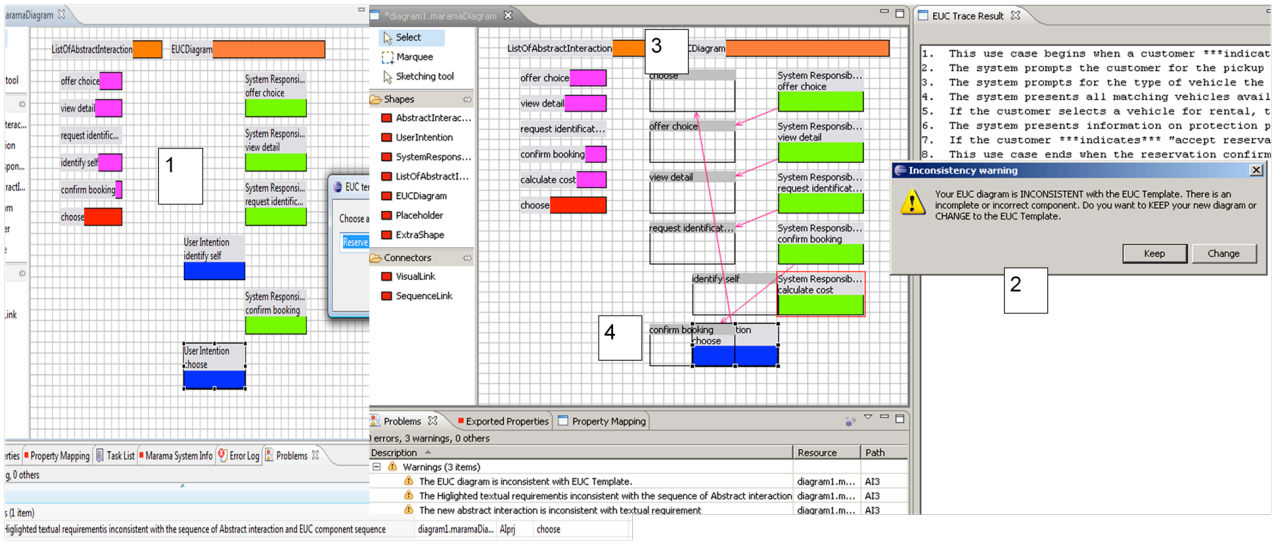

Figure 6. Visual differencing for detecting inconsistency, incompleteness and incorrectness using an EUC interaction pattern library template.

Here, EUC interaction pattern elements are shown as a set of grey elements adjacent to the extracted EUC model (3). Visual link "→" annotations connect corresponding elements in the extracted EUC and pattern. The tool is able to show errors such as wrong sequence ordering, redundancy, missing elements and existence of extra elements in the EUC model. Incorrect sequences are obvious via crossed links (e.g. the out of order "choose" abstract interaction). Unmatched items in the pattern template (e.g. "identify self") or in the extracted EUC (e.g. "calculate cost") are highlighted (4) (in this case juxtaposed to indicate the pattern element could sensibly replace the extracted element).

Based on the visualized 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. Our philosophy is to lessen the human effort and intervention in checking for potential errors but 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 [20].

5. ARCHITECTURE AND IMPLEMENTATION

We developed the MaramaAI toolset using our Marama meta-tools [21] and a number of specialised components for requirements extraction, analysis, comparison to pattern library and visual differencing. An outline of the tool's architecture is represented in Figure 7. 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 requirements are extracted from plain text documents (which themselves can be extracted from Word and PDF formats). This is done by using interaction phrase to abstract interaction mappings in our Abstract Interactions 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 associated EUC diagram (4).

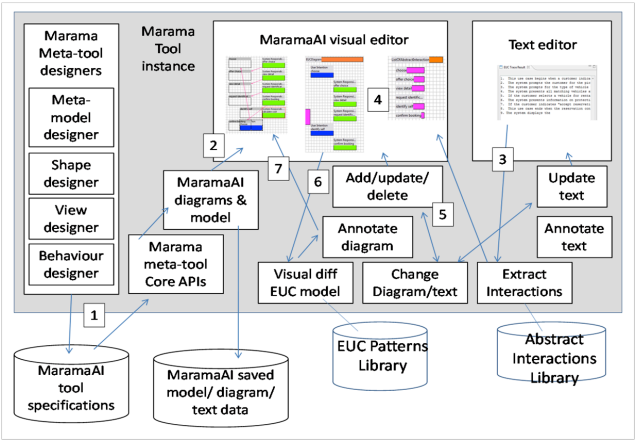

Figure 7. MaramaAI tool architecture.

We have mapped our EUC interaction pattern library approach, illustrated in Figure 2, to the consistency management framework proposed by [22]. The requirements engineer can make modifications to any of the representations supported by MaramaAI (5), including changing textual representation or adding, updating, resequencing 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 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 via 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 either to resolve the inconsistency by modifying components, tolerate the inconsistency (deferring for later attention) or indicate they wish to ignore the inconsistency. 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.

We implemented the visual diagramming interfaces of MaramaAI using the Marama metatool [21]. This supports rapid design and development of domain specific visual languages, and we used these facilities to develop the notations and editors for the abstract interaction listing, the EUC models and the EUC interaction patterns (the latter used in the visual differ). The meta-model and DSVL editors were 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 assist the user tracking and resolving inconsistencies. An event handler was implemented in Java to implement extraction of textual requirements into abstract interactions, and another to generate an EUC model from the abstract interaction. Two further event handlers are written in Java, one to perform a comparison of the EUC model to the pattern template library, and one to perform the visual differencing using Marama APIs to annotate the EUC diagram.

6. EVALUATION

In our prior work, we demonstrated that end users find manual extraction of EUCs difficult, time consuming and error prone [1] We also demonstrated that the algorithms we have used for abstract interaction and EUC extraction from natural language, produce much more accurate EUCs than manual extraction provides and far more rapidly. We wanted to demonstrate the effectiveness of our inconsistency, incompleteness and incorrectness detection approach along with the usability of our tool in supporting an engineer improve requirements quality with this approach. To this end we conducted an end user study to evaluate user perceptions of the tool and its application. Participants in the study were 11 software engineering post-graduate students, several of whom had previously worked in industry as developers and/or requirements engineers. All had some familiarity with EUC modeling, but were not experts in the approach. Each participant was given a brief tutorial on how to use the tool and some examples of how the tool manages consistency between the three requirement forms. They then undertook exercises to extract abstract interactions from textual requirements and map the abstract interactions to Essential Use

Cases. Further exercises on modifications to any of the requirements forms followed: adding and deleting elements, changing names and sequence ordering and observation of the resulting consistency management. Finally participants used the pattern library and visual differencing for further requirements checking and quality improvement.

Having familiarized themselves with the tool's capabilities, users completed a three part survey comprising: (1) a standard evaluation of user perceptions of usefulness, ease of use, ease of learning and satisfaction using a set of three questions addressing each of these characteristics; (2) a set of questions to determine user perceived strengths of selected dimensions of the cognitive dimensions of notations (CD) [23], a common approach for evaluating visual language environments – the questions being adapted from [15]; and (3) open ended questions related to improvements participants desired. For (1) and (2) a five part Likert scale was used for each question.

Figure 8 shows the results for the standard usability survey. For each characteristic, the results of each corresponding three question block were averaged to produce the results shown. The results are very positive, with strong agreement over the usefulness of the tool (over 80% strongly agree or agree on its usefulness), the ease of use (over 90%), ease of learning (over 90%) and satisfaction (over 80%). The small number of cases of disagreement over usefulness 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 available templates available in the EUC interaction pattern library (shown in answers to the open ended questions). However, overall these results are very encouraging, particularly given prior studies, our own and others, that suggest EUC modeling, while appealing to end users, has a large barrier to entry due to difficulty of use [17].

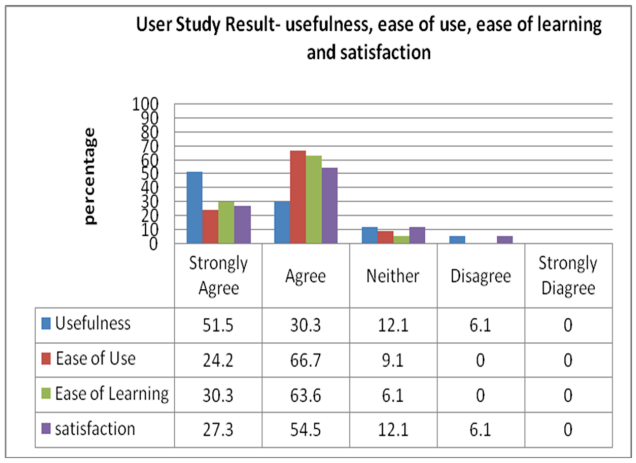

Figure 8. User study result-Usefulness, ease of use, ease of learning and satisfaction

The CD study allowed us to explore in more detail the reason for these user perceptions. Table 2 shows each of the dimensions we were interested in evaluating and the question addressing it. Table 3 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. The visibility and viscosity results demonstrate that users are comfortable with the tool and find it easy to make changes to the diagrams representing the various notational forms. The strong *closeness of mapping* rating and the relatively neutral *hard mental effort* and *error proneness* ratings point to the EUC notations being seen as relatively intuitive and understandable by our users. This is in strong contrast to the difficulty found by users in understanding and applying EUCs found in the prior studies. The consistency management and automated extraction support appears to be responsible for this, as demonstrated by the high ratings for visibility of dependencies, consistency of notations and progressive evaluation.

Table 2 Cognitive dimensions and questions evaluating them

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 3: Evaluation result for Cognitive Dimensions questions

Cognitive dimension	Strongly Disagree	Disagree	Neither	Agree	Strongly Agree
Visibility	0.0	0.0	0.0	90.9	9.1
Viscosity	0.0	9.1	9.1	72.7	18.2
Diffuseness	0.0	0.0	9.1	63.6	27.3
Hard-mental effort	9.1	27.3	45.5	18.2	0.0
Error-Proneness	0.0	54.5	45.5	0	0.0
Closeness of Mapping	0.0	9.1	9.1	72.7	9.1
Consistency	0.0	0.0	18.2	72.7	9.1
Hidden Dependencies	0.0	0.0	18.2	54.5	27.3
Progressive Evaluation	0.0	0.0	18.2	54.5	27.3
Premature Commitment	0.0	0.0	18.2	45.4	36.4

When it was introduced, the EUC semi-formal requirements modeling approach promised significant advantages in terms of quality enhancement and reusability [24]. In practice, however, the approach suffered from an inability for users to get the "right" level of abstraction when turning textual requirements into EUCs, plus difficulty in keeping EUC models consistent with textual requirements as requirements evolved. This led to poor performance of the methodology in practice and low uptake [1]. Combining the results of our user study with our prior efficacy results, it is apparent that, through appropriate tool support, the promised potential of the EUC approach is achievable.

7. RELATED WORK

The management of consistency and improvement in quality of requirements is an active area of research. Goknil et al. [25] 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 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 the requirement. The visualized result of either inferred relations or inconsistencies needs to be interpreted manually by the requirement engineer which can lead to errors [25]. Our work fills some of the gaps here as we cover consistency checking between multiple requirement artifacts and provide automation support for capturing textual requirement, generating Essential Use Cases and checking inconsistency besides providing visual support for detecting and resolving inconsistency, incompleteness and incorrectness problems.

Kroha et al. [26] investigated use of semantic web technology to check 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 [26]. This work does not, however, check for behavioral consistency as they cannot represent dynamic aspects of UML specifications in the ontology. By contrast, our work focuses on behavioral requirements, complementing their approach.

Graaf and Deursen [27] check for consistency between two types of behavioral specification: scenarios and state machines. They apply model transformations to generate state machines from scenarios. They then compare the generated state machines with manually developed state machines [27]. Some aspects of this work are similar as ours as we also compare the generated EUC model with our manually specified interaction library templates. However we differ in the representation model used and the purpose for matching the models. We also automate the checking and visualize the comparison results using visual differencing. We also support continued consistency management support with a natural language representation of requirements.

Nentwich et al. propose a repair framework for inconsistent distributed documents [28]. They generate interactive repairs from an input of 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 faced problems when the repair actions interacted with the grammar in a document, and also actions generated by other constraints [28]. Their approach also fails to identify a single inconsistency that may lead to other inconsistencies [28]. Our work overcomes this via our EUC interaction pattern library which better tackles the management of natural language requirements across a broad set of domains. In addition, our tool provides support for recognizing both single and multiple inconsistencies in requirement components. It also highlights and informs users via feedback and warning notification about affected components.

Mehra et al. apply visual differencing and diagram versioning and merging to support asynchronous collaborative diagramming [19]. Visual differencing allows users to interactively view and resolve differences between multiple diagram versions [19]. We have adopted this visual differencing approach to highlight differences between the generated EUC model and template. However, our focus is broader, supporting checks for inconsistency, incompleteness and incorrectness errors that exist in the generated model compared to the template, instead of just syntactic inconsistencies between diagram versions.

8. CONCLUSIONS

We have described a novel approach supporting requirements quality improvement via a combination of semi-formal model extraction from natural language specifications and analysis using an EUC interaction pattern library. Low-level inconsistency problems can be identified such as 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 to the Essential interaction pattern and to the "best practice" examples of EUC interaction pattern templates. A visual differencing technique highlights differences between the pattern template and EUC model. Modifications to EUC, abstract interaction and natural language requirements representations are also supported with consistency management support between the different representations. We evaluated our prototype tool from effectiveness and usability perspectives and the CD framework using an end user study. The results of this study are promising with most participants finding our tool to be useful for improving quality and managing consistency of requirements.

In future work we are developing an Essential User Interface (EUI) representation and a form based UI designer in order to visualize requirements as likely interface models as they captured. The generated UI could be used to show that requirements expressed by clients are consistent with the requirement engineer's view using this alternative visualization mechanism. Further possibilities include generation of alternative concrete interface forms from the abstract EUC model exploiting specific technology choices allowing an end to end rapid user interaction prototyping mechanism.

9. ACKNOWLEDGEMENT

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

10. REFERENCES

[1] Kamalrudin, M., Grundy, J. and Hosking, J., Tool Support for Essential Use Cases to Better Capture Software Requirements. In *Proceedings of the 25th IEEE/ACM International Conference on Automated Software Engineering*, Antwerp, Belgium, Sept 16-20 2010, ACM.

[2] Fabbrini, F., Fusani, M., Gnesi, S. and Lami, G., The linguistic approach to the natural language requirements quality: benefit of the use of an automatic tool. In *Proceedings of the 26th Annual NASA Goddard Software Engineering Workshop*, 2001, 97-105.

[3] Kamalrudin, M. Automated Software Tool Support for Checking the Inconsistency of Requirements. In *Proceedings of the 2009 IEEE/ACM International Conference on Automated Software Engineering*, Auckland,

New Zealand, Nov 16-20 2009, IEEE CS Press, pp. 693-697.

[4] Zowghi, D., and Gervasi, V. On the interplay between consistency, completeness, and correctness in requirements evolution. *Information and Software Technology 45* (14), November 2003, 993-1009.

[5] Spanoudakis and G, Zisman. A. Inconsistency Management in Software Engineering in "Handbook of Software Engineering and Knowledge Engineering.", vol 1, World Publishing, 2001, pp. 329-380.

[6] Nuseibeh, B., Easterbrook S. and Russo, A. Leveraging Inconsistency in Software Development. *Computer* 33 (4), 2000, 24-29.

[7] Satyajit, A., Hrushikesha, M. and George, C., Domain consistency in requirements specification. In *Proceedings of the Fifth International Conference on*, *Quality Software*, Melbourne, Australia, September 2005, IEEE CS Press, pp. 231-238.

[8] Kozlenkov, A. and Zisman, A., Are their Design Specifications Consistent with our Requirements?. In *Proceedings of 10th Anniversary IEEE Joint International Conference on Requirements Engineering*, Essen, Germany, September 2002, IEEE CS Press, pp. 145-156.

[9] Boyd, S., Zowghi, D. and Farroukh, A., Measuring the expressiveness of a constrained natural language: an empirical study. In *Proceedings of the 13 IEEE International Conference Requirements Engineering*, Paris, France, 2005, IEEE CS Press, pp. 339-349.

[10] Do, K., Method and Implementation for Consistency Verification of DEVS Model against User Requirement. In *Proceeding of the 10th International Conference on Advanced Communication Technology,* Gangwon-Do, Korea, 2008, pp. 400-404.

[11] Egyed, A., Scalable Consistency Checking Between Diagrams-The ViewIntegra Approach. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, San Diego, California, November 2001, IEEE CS Press, pp. 387.

[12] Denger, C., Berry, D.M. and Kamsties, E., Higher Quality Requirements Specifications through Natural Language Patterns. In *Proceedings of the IEEE International Conference on Software-Science, Technology & Engineering*, Herzlia, Israel, 2005, IEEE CS Press, pp. 80.

[13] Kamalrudin, M, Grundy, J and Hosking J. Managing consistency between textual requirements, abstract interactions and Essential Use Cases. In *Proceeding of the 34th Annual IEEE International Computer Software& Applications*, Seoul, Korea, July 2010, IEEE CS Press, pp. 327-336.

[14] 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 Longman, Inc, 1999.

[15] Kutar, M, Britton C., and Wilson. J, Cognitive Dimensions An Experience Report. In *Proceeding of the Twelfth Annual Meeting of the Psychology of Programming Interest Group, Memoria*, Cozenza Italy, 2000, pp. 81-98.

[16] Biddle, R., Noble, J. and Tempero, E. 2002. Essential use cases and responsibility in object-oriented development. In *Proceeding of the twenty-fifth Australasian conference on Computer science*, Melbourne, Victoria, Australia, 2002, ACM, pp. 7-16.

[17] Biddle, R, Noble J. and Tempero, E., Patterns for Essential Use Case Bodies, CRPIT '02 Proceedings of the 2002 conference on Pattern languages of programs, vol 13, Australian Computer Society, pp 85-98.

[18] Huzar, Z., Kuzniarz, L., Reggio, G. and Sourrouille, J.L. Consistency Problems in UML-Based Software Development. In *UML Modeling Languages and Applications*, 2005, 1-12.

[19] Mehra, A., Grundy, J and Hosking, J. A generic approach to Supporting Diagram Differencing and Merging for Collaborative Design. In *Proceedings of the 20th IEEE/ACM international Conference on Automated software engineering*, Long Beach, CA, USA, 2005, ACM, pp. 204 – 213.

[20] Ghose, A., Koliadis and G., Chueng, A. Process Discovery from Model and Text Artefacts *Services, In Proceeding of the 2007 IEEE Congress on Services*, Salt Lake City, UT ,July 2007, IEEE CS Press, pp. 167-174.

[21] Grundy, J.C., Hosking, J.G, Huh J. and, Li, N., Marama: an Eclipse meta-toolset for generating multi-view environments. In *Proceedings of the 2008 IEEE/ACM International Conference on Software Engineering*, Liepzig, Germany, May 2008, ACM, pp. 819-822.

[22] Nuseibeh, B., Easterbrook S., and Russo, A. Making inconsistency respectable in software development. *Journal of Systems and Software*, *58* (2), Sept 2001, pp. 171-180.

[23] Blackwell, A. and Green, T. A cognitive dimensions questionnaire optimised for users. In *Proceeding of the Twelfth Annual Meeting of the Psychology of Programming Interest Group*, Corigliano Calabro, Cosenza, Italy, 2000, pp. 137-152.

[24] Biddle, R., Noble, J., and Tempero E. 2002. Supporting Reusable Use Cases. *Lecture Notes in Computer Science,* Volume 2319, Springer-Verlag, 2002, pp. 135-138.

[25] Goknil, A., Kurtev, I., van den Berg, K. and Veldhuis, J.W. Semantics of trace relations in requirements models for consistency checking and inferencing, *Software and Systems Modeling*, 2009, pp. 1-24.

[26] Kroha, P., Janetzko, R. and Labra, J.E., Ontologies in Checking for Inconsistency of Requirements Specification. In *Proceeding of the Third International Conference on Advances in Semantic Processing,* Sliema, Malta, October 2009, IEEE CS Press, pp. 32-37.

[27] Graaf, B. and Deursen, A.V. Model-Driven Consistency Checking of Behavioural Specifications. In *Proceeding of the* F*ourth International Workshop on Model-Based Methodologies for Pervasive and Embedded Software*, Braga, Portugal, 2007, IEEE CS Press, pp. 115-126.

[28] Nentwich, C., Emmerich, W. and Finkelstein, A. Consistency management with repair actions. In *Proceedings of the 25th International Conference on Software Engineering*, Portland, Oregon, 2003, IEEE CS Press, 455-464.