LASR: A Tool for Large Scale Annotation of Software Requirements

Ishrar Hussain, Olga Ormandjieva, and Leila Kosseim
Department of Computer Science and Software Engineering
Concordia University
Montreal, Quebec, Canada
{h_hussa, ormandj, kosseim}@cse.concordia.ca

***Abstract*—Annotation of software requirements documents is performed by experts during the requirements analysis phase to extract crucial knowledge from informally written textual requirements. Different annotation tasks target the extraction of different types of information and require the availability of experts specialized in the field. Large scale annotation tasks require multiple experts where the limited number of experts can make the tasks overwhelming and very costly without proper tool support. In this paper, we present our annotation tool, LASR, that can aid the tasks of requirements analysis by attaining more accurate annotations. Our evaluation of the tool demonstrate that the annotation data collected by LASR from the trained non-experts can help compute gold-standard annotations that strongly agree with the true gold-standards set by the experts, and therefore eliminate the need of conducting costly adjudication sessions for large scale annotation work.**

***Keywords*-Software Requirements Analysis; Requirements Annotation; Linguistic Annotation Tool.**

I. INTRODUCTION

A. Annotation of Software Requirements

Software requirements annotation involves annotating different parts of the software requirements document to indicate what classes of requirements they contain, or, which software engineering artifacts are present (e.g. domain entities, data-attributes etc.), or, any other classes of information vital to the software project. For example, Fig. 1 shows an extract from a software requirements document and how an annotator has chosen to annotate its sentences with different annotation labels. Here, the requirements sentences are to be annotated into four different classes:

(i) Functional Requirement: A software requirement that expresses the required behavior of the system.

(ii) Non-Functional Requirement: A software requirement that expresses the quality requirements and the constraints over the related behavior of the system.

(iii) Ambiguous Requirement: A software requirement that can be interpreted in more than one way by the annotator (i.e. the requirements analyst).

(iv) Noise: Any sentence that does not express any of the above types of software requirement.

Software requirements can be further annotated according to different needs of information that are to be extracted about the software to be developed.

Extract from a requirements document

...The following use case describes approving a budget. First, the user navigates to the budget overview page. The system then displays the budget overview with editable budget attributes. System presets some of the budget attributes. User edits the budget attributes and sets the status as "Approved". All the mandatory attributes cannot be empty and the budget amounts cannot be negative. User finally saves the budget. ...

Requirement Sentence	Annotation Label
The following use case describes approving a budget.	Noise
First, the user navigates to the budget overview page.	Functional
The system then displays the budget overview with editable budget attributes.	Functional
System presets some of the budget attributes.	Functional
User edits the budget attributes and sets the status as "Approved".	Functional
All the mandatory attributes cannot be empty and the budget amounts cannot be negative.	Non-Functional
User finally saves the budget.	Functional

Figure 1. Example of Sentence-level Annotation of Software Requirements.

B. Motivation and Research Objective

In the early phases of the software development lifecycle, software requirements are essentially captured in unrestricted natural language without any formalization, so that it can be easily conveyable between the clients (and/or the potential users) and the technical people (analysts, developers, managers and others) [1]. After the elicitation of software requirements, the tasks of requirements analysis usually involve cumbersome manipulation and organization of a large pool of textual requirements. Being written in unrestricted natural language, these textual requirements are often found to be corrupted with ambiguity that an expert has to manually identify and resolve [2]. The documents containing the textual requirements can also be either unstructured or of varying structures which demands additional effort from an expert to manually extract crucial knowledge about the software. For example, sentences describing non-functional requirements are often found embedded in paragraphs containing functional requirements that an expert often has to manually organize by separating the non-functional requirements from the functional ones [3].

Thus, the annotation of software requirements is a crucial activity performed by experts to deal with informally written requirements during the requirements specification phase [4]. Reuse of requirements documents also require annotating its parts following a standard of requirements taxonomy [5].

However, different types of requirements annotation tasks are targeted to extract different kinds of information about the software from its requirements. Thus, to have many experts on different types requirements annotation

tasks always available for all projects adds to the overall project costs.

Again, requirements annotation is also necessary to build annotated sets of documents (i.e. annotated corpora) that are used in numerous recent researches [6] that attempt to learn the behavior of human expertise behind different requirements analysis tasks and automate these tasks by using supervised or semi- supervised learning techniques. For example, our umbrella project, READ-COSMIC [7], requires annotated corpora to train and test our supervised text miners that are used for measuring the functional size of software. To build such annotated corpora for these researches would ideally require many human experts to manually annotate a large number of requirements instances. This would not only be highly expensive, but also overwhelming for a limited number of experts.

Thus, the objective of the research presented in this paper is to identify a unique set of features for a requirements annotation tool that would—

(g1) Support any type of requirements annotation tasks.

(g2) Support building large pools of annotated corpora for statistical data analysis.

(g3) Improve the overall process of requirements annotation by attaining accurate annotations with non-experts.

To achieve this, we developed our annotation tool, called "LASR", with the aim to satisfy the above goals. In this short paper, we will present the experiments which we conducted to validate how well LASR satisfies goal *g3* above. Let us first describe LASR briefly.

II. LASR: Live Annotation of Software Requirements

We developed *LASR* (*Live Annotation of Software Requirements*) to aid the collection of annotated corpora and the generation of training/testing datasets required by the supervised learning systems related to our work [7]. LASR is a Web-based application that provides a rich graphical user interface allowing quick navigation and control during the annotation tasks. It uses a client-server architecture at the highest-level of the logical view. On the server-side, it implements a three-tier-architecture, comprising of the Presentation, Application and Services layers. The application layer then further implements the model-view-controller architecture, via the CakePHP framework. Figure 2 shows LASR's architecture in details.

Here, the *Requirements Repository* at the backend holds the requirements documents contributed by its users. The *Instance Extractor* module of LASR is equipped with lightweight NLP-based tools, e.g. a sentence delimiter and a noun-phrase chunker, that can automatically extract requirements instances at the levels of passages, sentences and noun-phrases from the requirements documents and save them to the backend. *Annotation Templates* define the annotation work to be performed at a particular level of requirements instance (e.g. at the sentence level or noun-phrase level). The templates are stored as XML files at the backend file-system, and contains configuration details on the annotation interface as well, making the interface customizable by the project manager.

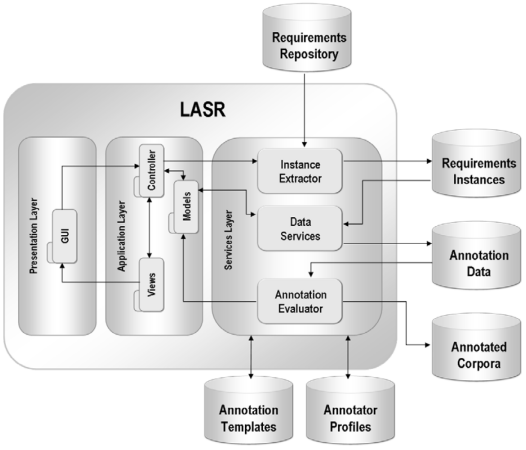

Figure 2. Architecture of LASR.

III. Experiments & Results

To validate our goal *g3*, we ran annotation experiments on LASR at the sentence-level of software requirements (as shown in section I.A) and compared its accuracy to that obtained with manual annotation, without any tool support.

A. The Corpus

For these experiments, our corpus was composed of six requirements documents belonging to three different problem domains. They were collected from both the industry and academia. Some statistics about these documents are presented in Table I.

TABLE I
Documents used in the experiments

Doc. ID	Doc. Title	Source	Problem Domain	Total Sentences Extracted After Preprocessing
D1	*(undisclosed)*	SAP Labs, Montreal, Canada	Business	15
D2	*(undisclosed)*	SAP Labs, Montreal, Canada	Business	101
D3	Course Registration System	Concordia University	Academic (Private)	179
D4	IEEE Montreal Website	Concordia University	Web (Public)	467
D5	*(undisclosed)*	SAP Labs, Montreal, Canada	Business	7
D6	*(undisclosed)*	SAP Labs, Montreal, Canada	Business	89

Here, we extracted the sentences from only those sections of the documents that held textual user requirements.

B. The Annotators

We had two groups of annotators and one expert (in requirements annotation) annotating the above documents. One group (G1) consisting of four graduate students of the Master of Computer Science program who were trained to annotate software requirements documents manually. The expert led the training of the annotators, and also participated with them in the manual annotation experiment.

The other group (G2) consisted of 26 undergraduate students of software engineering,. They were introduced to requirements annotation (through lectures and handbooks), but were not thoroughly trained. No prior tests were conducted to verify their knowledge, before the experiment was executed. However, they were all trained via class tutorials to work on LASR's user interface as annotators.

We designed the experiments so that G1 annotated the requirements documents manually, and G2 used LASR to annotate the same documents.

C. Results

A total of 858 sentences (from the six documents of Table I) were annotated during this experiment. We identified the expert's annotation as the true gold-standard to compare all other annotations made during the experiments, both manually and by LASR. The distribution of the true gold-standard annotations of our corpus, as annotated by the expert, is shown in Fig. 3.

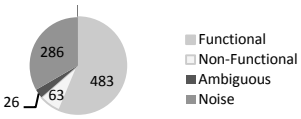

Figure 3. Distribution of the true gold-standard annotations (as annotated by the expert) in our corpus.

1) Manual Annotation: The annotators of G1 (excluding the expert) performed the annotation manually on four documents only, instead of the six, for a total of 742 sentences. The gold-standard annotations were computed by following the majority voting model. Whenever a gold-standard annotation could not be resolved, adjudication was performed by the participation of the annotators of G1. Here, we evaluated the performance of the annotators in terms of Cohen's kappa [8], showing the degree of agreement of the gold-standard annotations, computed collectively from the four annotators, with the annotations of the expert (the true-gold-standards). This showed a high degree of agreement (Kappa = 0.83187). This result represents the best-case scenario, where all four annotators of G1 were properly trained and the gold-standard annotations were computed after holding meticulous adjudication sessions with their participation to resolve their points of disagreements.

2) Annotation with LASR: The annotators of G2 used LASR to perform annotation of all the six documents for a total of 858 sentences. We computed the gold-standard annotations manually using the simple majority voting rule (we call this method, *M1*). We found that the computed gold-standard annotations this way moderately agree (Kappa = 0.72396) with those submitted by the expert. Figure 4 shows the corpus distribution in this case.

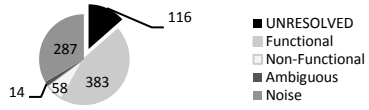

Figure 4. Distribution of the gold-standard annotations computed manually, based on majority voting (*M1*).

Here, we found that the method, *M1*, could not resolve the gold-standards for 116 of the instances, indicating a high degree of disagreements for those instances. The standard method to resolve this issue is to run adjudication sessions for all unresolved instances with the participation of all annotators. Unfortunately, this method is costly especially with real annotation tasks performed over larger corpora.

To address this problem, the first option that LASR introduces, is to compute the gold-standard annotations automatically using the levels of confidence entered by the annotators. We call this method of computing gold-standard annotations *M2* for our experiments. Here, LASR tries to compute the gold-standard annotation for each of the annotated instances, by first assigning a custom score to each of the annotation labels based on the level of confidence submitted by the annotators. Thus, the annotation label with the highest score, and that is also greater than some threshold (0.5[1] in our case), is selected as the gold-standard annotation. LASR uses the formula below to calculate the score for each possible annotation. It shows that if m_i annotators have annotated an instance i, and the class c is one of the possible class labels for annotating the instance i, then the score of class c for the instance i is —

$$Score(c \mid i) = \frac{1}{m_i} \times \sum_{x=1}^{m_i} (conf_{x,c,i}) \qquad (1)$$

Thus, the final score of an annotation class c for an instance, i, is the arithmetic average of all the $conf_{x,c,i}$ values submitted by each annotator, x. The confidence of an annotator x, denoted by $conf_{x,c,i}$ in the formula (1), is equal to one of values of {0.1, 0.4, 0.7, 1.0} that is according to the level of confidence chosen by the annotator x, while annotating the instance i as class c. And, for all those classes, c', that are not chosen the annotator x for the instance i, $conf_{x,c',i}$ will be equal to 0. Thus, $0 \leq conf_{x,c,i} \leq 1$.

We used LASR to compute the gold-standard annotation labels according to *M2*. Their distribution is shown in Fig. 5. It shows that the computed gold-standard annotations now agrees highly (Kappa = 0.81184) with those submitted by our expert. However, there still remain 20 instances for which the gold-standards could not be resolved.

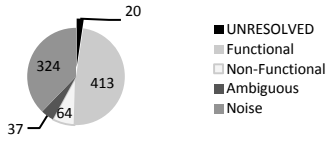

Figure 5. Distribution of the gold-standard annotations computed by LASR, using annotators' confidence levels (*M2*).

LASR also provides a second option to compute gold-standard annotation. Here, the expert first seeds the

[1] LASR requires the annotators to attach fuzzy levels of their confidence to each of their annotations. The level of annotator's confidence is collected as 4-value ratings that are then translated into fuzzy numeric values, all as positive real numbers≤1, having equal intervals, and none being 0.5 or 0. We wanted no annotation to be ignored because of a zero weight or be indecisive because of a 0.5 weight on the confidence level.

annotation data by setting true gold-standard annotations for at least a small portion of the un-annotated corpora. LASR recommends seeding by annotating at least 10% of the un-annotated instances randomly. The seeded annotations are regarded as the true gold-standard annotations; and LASR then measures the skill level, S_x, of each annotator, x, as the ratio of his/her annotations agreeing with the true gold-standard annotations. For example, if the annotations of an annotator, x, agrees with the seeded gold-standard annotations 60% of the times, then the skill level, S_x, would be 0.6. Thus, LASR uses a modified version of the formula (1) as below to calculate the score for selecting the gold-standard annotation label for each instance—

$$Score(c \mid i) = \sum_{x=1}^{m_i} (conf_{x,c,i} \times S_x) \qquad (2)$$

Thus, the annotation label c that achieves the highest score in terms of the above formula (2) is selected as the gold-standard annotation for an instance i. We call this method of computing gold-standard annotations *M3* for our experiments. We now use LASR to compute the gold-standard annotation labels according to *M3*. Their distribution is shown in Fig. 6. It shows that the computed gold-standards annotations now have a very high degree of agreement (Kappa = 0.86043) with those submitted by our expert. Moreover, there remained no instances, where their gold-standards are unresolved, eliminating the need for running costly adjudication sessions.

Figure. 6. Distribution of the gold-standard annotations computed by LASR, using annotators' confidence levels and skill levels (*M3*).

Figure 7 summarizes the results of our annotation experiments, showing the quality of the computed gold-standard annotations (based on the annotations submitted by G2) for *M1*, *M2* and *M3*, in terms of their degrees of agreement (in Kappa) with the true gold-standard annotations chosen by the expert. Fig. 7 also compares these results to that of the gold-standard annotations after G1 performed the task manually.

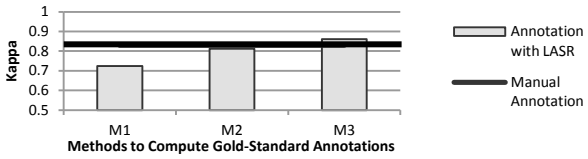

Figure 7. Quality of the different gold-standard annotations in terms of their agreements (in Kappa) with the true gold-standard annotations

Our experiment with *M3* shows that LASR was able to weight the annotations based on the scores of formula (2) accordingly, using the levels of skill of the annotators of G2 and their levels of confidence, and, thus, selected the gold-standards that agreed the most with the true gold-standards, as shown in Fig. 7. This indicates that LASR automatically extracted gold-standard annotations that are reliable enough, even when the group of annotators was not fully trained.

IV. Conclusion

In this paper, we discussed how a linguistic annotation tool can effectively aid the annotation tasks of software requirements. We presented our annotation tool, LASR, and showed how it helped a group of annotators with minimum training to annotate software requirements accurately.

The unique features of LASR did not only help in attaining more accurate annotations, but also helped eliminating the need of running adjudication sessions to resolve disagreement among the annotators, and, thus, reducing the cost of large scale annotation. It improved the idea of crowd-sourcing by introducing the method of expert seeding of the true gold-standard annotations for a portion of the corpora, and thus, allowing real-time evaluation of the skills of annotators.

For our future work, we intend to further experiment on the usability of LASR, and study if it can help minimize the annotators' effort for different requirements annotation tasks. Also, we would analyze the impact on our results for having different numbers of annotators. The data collected from our experiments on LASR are currently also being used for our umbrella project, READ-COSMIC [7], that explores the linguistic features for functional size measurement.

Acknowledgment

We thank SAP Labs, Montreal, Canada, for providing the annonymized software requirements documents for our experiments. We also thank the anonymous reviewers for their comments on an earlier version of this paper.

References

[1] Leffingwell, D., & Widrig, D. (2003). Managing Software Requirements: A Use Case Approach. Pearson Education.

[2] Meyer, B. (1985). On Formalism in Specifications. IEEE Software, 2, 6-26.

[3] Hussain, I., Kosseim, L., & Ormandjieva, O. (2008). Using Linguistic Knowledge to Classify Non-functional Requirements in SRS documents. In Natural Language and Information Systems, LNCS 5039, pp. 287-298. Germany: Springer-Verlag.

[4] Ko, Y., Park, S., Seo, J., & Choi, S. (2007). Using classification techniques for informal requirements in the requirements analysis-supporting system, Information and Software Technology Journal, 49, 1128-1140.

[5] Eriksson, M., Börstler, J. & Kjell, B. (2009). Managing requirements specifications for product lines – An approach and industry case study, The Journal of Systems and Software, 82, 435–447.

[6] Casamayor, A., Godoy, D., Campo, M. (2011). Mining textual requirements to assist architectural software design: a state of the art review, Artificial Intelligence Review, Springer Netherlands, 1-19.

[7] Hussain, I., Kosseim, L., & Ormandjieva, O. (2010). Towards Approximating COSMIC Functional Size from User Requirements in Agile Development Processes Using Text Mining. In Natural Language Processing and Information Systems, LNCS 6177, pp. 80-91. Germany: Springer-Verlag.

[8] Cohen, J. (1960). A Coefficient of Agreement for Nominal Scales. Journal of Educational and Psychological Measurement, 20, 37-46.