Automated Retrieval of Graphical User Interface Prototypes from Natural Language Requirements

Kristian Kolthoff[1], Christian Bartelt[1] and Simone Paolo Ponzetto[2]

[1] Institute for Enterprise Systems, University of Mannheim, Mannheim, Germany
{kolthoff,bartelt}@es.uni-mannheim.de
[2] Data and Web Science Group, University of Mannheim, Mannheim, Germany
simone@informatik.uni-mannheim.de

Abstract. High-fidelity Graphical User Interface (GUI) prototyping represents a suitable approach for allowing to clarify and refine requirements elicitated from customers. In particular, GUI prototypes can facilitate to mitigate and reduce misunderstandings between customers and developers, which may occur due to the ambiguity and vagueness of informal Natural Language (NL). However, employing high-fidelity GUI prototypes is more time-consuming and expensive compared to other simpler GUI prototyping methods. In this work, we propose a system that automatically processes Natural Language Requirements (NLR) and retrieves fitting GUI prototypes from a semi-automatically created large-scale GUI repository for mobile applications. We extract several text segments from the GUI hierarchy data to obtain textual representations for the GUIs. To achieve ad-hoc GUI retrieval from NLR, we adopt multiple Information Retrieval (IR) approaches and Automatic Query Expansion (AQE) techniques. We provide an extensive and systematic evaluation of the applied IR and AQE approaches for their effectiveness in terms of GUI retrieval relevance on a manually annotated dataset of NLR in the form of search queries and User Stories (US). We found that our GUI retrieval performs well in the conducted experiments and discuss the results.

Keywords: Automatic Prototyping of Graphical User Interfaces (GUIs), GUI Retrieval, GUI Prototypes from Natural Language Requirements

1 Introduction

Effective requirements elicitation techniques play a vital role in early development stages [18], in order to mitigate or eliminate misunderstandings of requirements between customers and developers, which might occur due to the ambiguity and vagueness inevitably encompassed in Natural Language (NL) communication [4]. GUI prototyping poses a meaningful technique to visualize the developers' understanding of the requirements and enable their verification by the customer as a tangible artifact. Moreover, GUI prototypes can provide the foundation for incorporating the customer early into the application development and lead to productive discussions and clarification of requirements [16].

In this work, we propose an ad-hoc GUI retrieval approach that is based on a semi-automatically created large-scale GUI repository for mobile apps. Kolthoff et al. [12] showed how such a GUI retrieval system can be useful to support rapid prototyping and it could be used as part of a virtual prototyping assistant [11].

2 Approach: GUI2R

The main goal of our *GUI2R* approach is to retrieve matching GUI prototypes for a NL query provided by a user. In order to achieve that, we employ Natural Language Processing (NLP) and Information Retrieval (IR) techniques to compute a ranking over a large-scale GUI prototype repository for the given NL query. In addition, we experiment with various Automatic Query Expansion (AQE) techniques to tackle the vocabulary mismatch problem [13]. In our approach, we employ a GUI repository of mobile applications (Android). Fig. 1 shows an overview of the system architecture of *GUI2R*, which in general follows the extended Boolean model [13]. First, a user specifies a NLR in the form of a search query or in the structured form of a US. The input is processed *(A)* by a NLR parser that detects US and extracts only specific parts for further processing. Subsequently, a pipeline of text preprocessing techniques is applied on the NL input. As a foundation for the GUI repository, we employ *(B)* the large-scale mobile app GUI dataset *Rico* [7]. This dataset consists of Android apps crawled from Google Play. To make the GUI prototypes searchable for NL input, we extract particular text segments from the corresponding GUI hierarchy data and represent the GUI prototypes as text documents. Afterwards, *(C)* an inverted index is computed from the GUI text documents and applied to match GUI documents that contain at least a single query term. The matched GUI documents are then scored by a retrieval model and the top-ranked documents are used for AQE to compute (D) the final ranking of the GUI prototypes. In the following, we describe the individual components of *GUI2R* in more detail.

2.1 NLR Parsing and Preprocessing

We employ several text preprocessing methods on the NL input. First, we lowercase the NL input and apply tokenization. Tokens are then excluded by several filters: We remove basic English stopwords, words comprising numeric or non-ASCII characters and out-of-vocabulary words based on a dictionary derived from the textual representation of the GUIs. We initially apply our US parser that is based on pattern matching to detect US and extract the *user-role*, *user-task* and *user-goal* from the US template (based on the Connextra format). From the parsed US, we only use the *user-task* description as NL input to our GUI retrieval system, but apply previously discussed preprocessing steps beforehand.

2.2 GUI Repository and Preprocessing

Recent research on data-driven design published several GUI datasets suitable for our retrieval system such as *ReDraw* [14], *ERICA* [8] and *Rico* [7]. All of

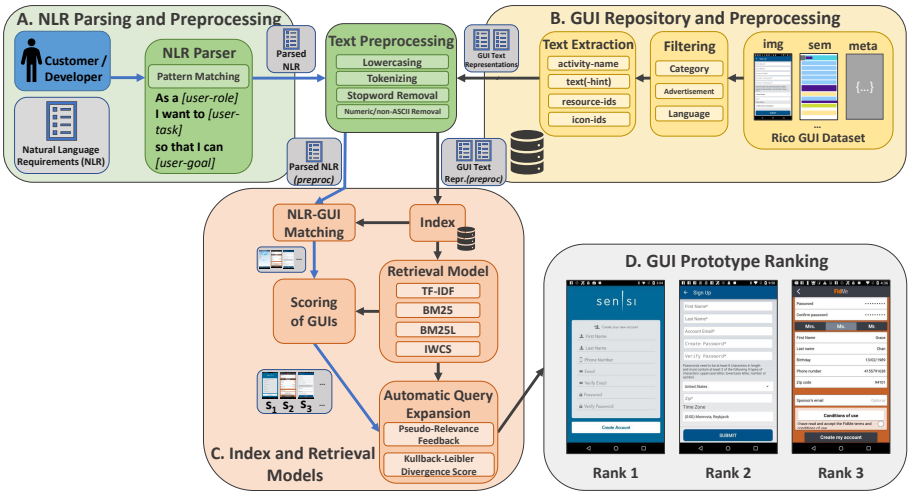

Fig. 1: Overview of *GUI2R* with (A) NLR parsing of US and text preprocessing, (B) GUI repository and its preprocessing, (C) Matching of NLR and GUIs with index and GUI scoring with multiple retrieval models and (D) final GUI ranking

these GUI datasets are gathered from mobile applications crawled from the app store Google Play. We decided to use *Rico* and the reasons for employing *Rico* as our GUI repository are manifold: *(i)* the large scale, making a retrieval system particularly valuable, *(ii)* the wide spectrum and diversity of mined applications available for retrieval, covering potentially many reusable GUIs and *(iii)* the provided rich textual information, including component identifiers and semantically tagged GUI components. *Rico* mines GUI screenshots, GUI hierarchy data, application meta data and interaction traces with both human-based and automatic exploration techniques and constitutes the largest design dataset of the discussed ones with 72,219 GUIs collected from 9,772 unique Android apps.

Since *Rico* crawls applications and extracts GUIs partly in an automatic fashion, incorporating noisy GUIs in the dataset is inevitable. First, (1) we filter all GUIs that belong to applications of the entertainment category, since we are not interested in game GUIs. Second, (2) we remove GUIs covered with advertisement overlay screens by checking for particular patterns of the component labels in the semantically tagged GUI document. Third, (3) we apply language detection on the extracted text segments in order to remove non-English GUIs from the repository. To achieve that, we employ a language detection framework that computes language probabilities by accumulating character-level n-gram spelling feature probabilities [17].

To enable NL-based search queries on the mobile app GUI repository, we first require to represent the GUIs as text documents. From the GUI hierarchy data provided in *Rico*, we extract several text segments through XPath expressions. We extract text from all components that are explicitly marked as `text`

or `text-hint` and displayed to the user. In addition, we extract the full activity name of the GUI and the resource identifier of each individual GUI component. Developers often provide semantically rich descriptive names for their activities and GUI component identifiers, thus we consider them as a valuable resource for retrieval. To make these special strings such as *"com.sample.sens.register.-CreateNewAccountActivity"* searchable, we apply a pipeline of various tokenizers. First, we apply punctuation, basic camel case and snake case tokenization. On top, we use a custom probabilistic tokenizer based on English Wikipedia unigram frequencies to split remaining concatenated words. On the resulting tokens, we apply a specially created stopword list to remove non-descriptive general terms (e.g. *"com"*, *"main"* and *"activity"*). From the semantically tagged GUI representation, we extract the textual descriptions of the detected icons (for example *"add"* and *"search"*) since they often provide descriptive terms that may similarly be used in NL queries. The text segments are preprocessed identically to the query and represent the GUI as a text document as the basis for GUI retrieval.

2.3 Information Retrieval Models

To retrieve matching GUIs from NL queries from our GUI repository, we adopt retrieval methods that have a long history in IR research [13]. In particular, we employ TF-IDF, BM25 and BM25L [15]. These models provide a strong baseline to many other specific IR tasks and have shown their effectiveness in other domains before. In addition, we evaluate a more recent method that exploits TF-IDF weighted pre-trained dense word embeddings (based on 300-dimensional *word2vec* embeddings) for similarity scoring (IWCS) [9]. Another way to enhance the retrieval performance of IR systems is the introduction of AQE techniques to tackle the vocabulary mismatch problem through Pseudo-Relevance Feedback (PRF) [13]. Many expansion candidate scoring methods based on PRF have been proposed [1]. These methods follow a similar underlying notion. First, a ranking over the terms contained in the relevant documents D_R (top-k documents initially retrieved by a base model) is computed. Here, terms that are special for the relevant documents D_R and distinguish them from the rest of the document collection D_C should receive a higher ranking score. This can be achieved through comparing the term distributions between the D_R and D_C documents. The initial user query is then expanded with the top-n words from the ranked candidate terms. For our retrieval system, we compute the Kullback-Leibler Divergence (KLD) score [5] for each term $t \in D_R$ as

$$Score_{KLD}(t) = p(t|D_R) \cdot \log \frac{p(t|D_R)}{p(t|D_C)}$$

with $p(t|D_R)$ and $p(t|D_C)$ being the probability of term t occurring in D_R and D_C, respectively. The probabilities are computed as the Maximum Likelihood Estimates i.e. $p(t|D_X) = \frac{f_{t,x}}{|D_X|}$. We decided to apply and evaluate the KLD score in our experiments since it showed its effectiveness compared to other scoring methods before [5]. We also evaluated two other variants that include the KLD

Table 1: Evaluation dataset overview and examples for the three different NLR

#	NLR Type	Size	Examples
(1)	KB Queries	30	**(1)** "daily log" **(2)** "watch video" **(3)** "image blog with search" **(4)** "export data" **(5)** "select clothing size between s and xl" **(6)** "select my age" **(7)** "image grid" **(8)** "new price old price product" **(9)** "show training statistics"
(2)	US (int.)	20	**(1)** "As a user I want to see the product price, product image and product description" **(2)** "As a user I want to choose my favorite language" **(3)** "As a user I want to see the number of votes of a post **(4)** "As a user I want to create a new account"
(3)	US (ext.)	10	**(1)** "As an OlderPerson, I want to maintain my contact list in my phone." **(2)** "As a user, I want to be able to search any dataset published and publicly accessible by their title and metadata, So that I can find the datasets I'm interested in." **(3)** "As a User I want to set my own username, So that my data is more easily discoverable."

score as a weight for the expanded terms in the retrieval model to control their influence and by computing expansion terms for each text segment separately.

3 Experimental Evaluation

In order to evaluate the proposed approach, we investigate two research questions that relate to the retrieval performance of the discussed IR and AQE models. In the following, we describe our evaluation dataset, the annotation schema, the employed evaluation metrics and discuss the obtained evaluation results. In particular, we investigate the following research questions:

- $\mathbf{RQ}_1$: Are traditional Information Retrieval (TF-IDF, Okapi BM25, BM25L) and more modern scoring functions (IWCS) suitable for GUI prototype retrieval from Natural Language Requirements (NLR)? Which method performs best for GUI retrieval from NLR?
- $\mathbf{RQ}_2$: Can pseudo-relevance feedback methods based on the Kullback-Leibler divergence score improve the retrieval performance using Okapi BM25 as a base model? Which AQE method performs best for GUI retrieval from NLR?

3.1 Experimental Setup

In order to evaluate the proposed research questions, we created a requirement collection consisting of 60 Natural Language Requirements (NLR), since there is no evaluation dataset available for evaluating the GUI retrieval systems performance. This dataset provides the foundation for our evaluation and is separated into three sub-datasets: (1) Keyword-based search queries that represent the typical format for conducting searches with 30 examples, (2) User Stories (US) (internal) that we created to investigate different US types and their application

in our GUI retrieval approach with 20 examples and (3) User Stories (US) (external) that we gathered from an external resource with 10 examples [6]. Table 1 shows an overview of the evaluation dataset and provides concrete examples for all three sub-datasets. For the keyword-based search queries sub-dataset (1), we attempted to include many diverse topics from rather broad queries such as (1.1) requiring daily log functionality and (1.2) requiring functionality to watch a video to more specific queries such as (1.5) requiring a particular clothing size selection range. For the internal User Story sub-dataset (2), we included US that represent typically reusable requirements and occur among many applications such as (2.2) requesting functionality to choose the favorite language or (2.4) requiring functionality to create a new account, but also US that are more specific to a particular domain and difficult such as (2.1) containing many details. For the external User Story sub-dataset (3), we employed a publicly available US requirement dataset [6] and gathered 10 US that are related to GUIs from two applications (*openspending* and *alfred*). These requirements are generally more specific and custom to their particular application such as example (3.2) requiring a dataset search functionality with very specific search parameters. To evaluate the retrieval performance in terms of relevancy of the returned GUI prototypes and since there is no goldstandard available for this particular problem, we annotated the retrieval results manually. We annotated the top-k retrieved GUI prototypes for all requirements from our evaluation dataset. In our experiments in particular, we retrieved and annotated the top-15 GUI prototypes for each requirement and method ($k = 15$). For a particular evaluation requirement, we annotated each retrieved GUI prototype on a relevancy scale of 0 (*not relevant*), 1 (*related*), 2 (*relevant*) through a web-based evaluation application. Finally, we computed the following standard IR metrics: Precision ($P@k$), Average Precision (AP) and Normalized Discounted Cumulative Gain ($NDCG@k$).

3.2 Results and Discussion

The evaluation results for our different experiments are shown in Table 2. For our first experiment ($\mathbf{RQ}_1$), we observe that BM25 outperforms all other evaluated IR models by a large margin for datasets (1) and (2), and only for dataset (3) TF-IDF outperforms all other models. IWCS can outperform TF-IDF on the search query dataset (1) but performs worse on both US datasets (2) and (3). During the annotation of the results, we observed some typical retrieval errors. GUI prototypes with an opened menu overlapping most of the screen were often ranked among the top-15 results since the underlying GUI contained relevant text that was not marked as non-visible. We also observed GUIs that were represented properly as textual documents, however, have erroneous GUI screenshots due to GUI capturing errors. Often, semantic retrieval errors occurred, for example, *login* screens which are retrieved for requirement (1.1) *"daily log"*.

For our second experiment ($\mathbf{RQ}_2$), we observe that BM25-PRF (w) and BM25-PRF (cw) outperform the BM25 model for most of the cases, however, often only on small margins. During the annotation, we observed that the base model performance could be improved especially for requirements that are less

Table 2: Evaluation results overview of the different experiments

	(1) Search queries					**(2) User Stories (int.)**					**(3) User Stories (ext.)**				
	P@1	*P*@5	*P*@15	*AP*	*N*@15	*P*@1	*P*@5	*P*@15	*AP*	*N*@15	*P*@1	*P*@5	*P*@15	*AP*	*N*@15
TF-IDF	.467	.427	.344	.496	.763	.500	.430	.407	.530	.777	**.200**	**.240**	**.193**	**.376**	**.532**
BM25	**.767**	**.633**	**.513**	**.710**	**.902**	**.600**	**.580**	**.503**	**.676**	**.839**	.000	.180	.167	.207	.493
BM25L	.333	.320	.256	.417	.714	.400	.310	.300	.398	.752	.100	.140	.153	.196	.412
IWCS	.600	.507	.427	.608	.825	.500	.400	.383	.518	.747	.100	.140	.087	.179	.336
BM25	**.767**	.633	.513	.710	.902	.600	.580	.503	**.676**	**.839**	.000	.180	.167	.207	.493
+PRF	.667	.647	.509	.687	.901	.650	.510	.460	.571	.837	**.400**	.180	.147	**.331**	.494
+PRF(c)	.633	.647	.484	.670	.887	.500	.550	.487	.601	.829	.300	.200	.160	.315	**.541**
+PRF(w)	.733	**.673**	**.527**	**.718**	**.905**	**.650**	.550	.543	.637	.823	.200	**.220**	.160	.257	.495
+PRF(cw)	.600	.673	.493	.672	.892	.650	**.590**	**.550**	.656	.831	.300	.200	**.180**	.264	.520

ambiguous and where it is in general easier to find matching GUIs for. For example, for requirements such as *"login"* or requirement (2.4) requesting functionality for creating a new account, the AQE method could filter out some incorrect GUIs by extracting relevant expansion terms from the top-ranked results.

4 Related Work

Guigle [3] automatically crawls and extracts GUI screenshots and GUI hierarchy data from Android apps harvested from Google Play [14]. Their approach indexes multiple parts of the hierarchy such as the app name, screen color, GUI component text and type and employs a basic Boolean query language to quickly retrieve relevant GUIs. However, our GUI retrieval system *GUI2R* particularly focuses on customer-friendly NLR input, proposes are more sophisticated retrieval architecture including IR methods based on word embeddings and AQE techniques and provides an in-depth evaluation of these methods. In contrast, *Swire* [10] and *GUIFetch* [2] enable mobile app GUI retrieval not through simple NL input, however, using basic Android apps or hand-drawn sketches. In particular, *Swire* employs a neural network-based joint embedding space between the GUI screenshots and the hand-drawn sketches for retrieval, whereas *GUIFetch* computes similarities between GUIs to rank applications based on an app sketch.

5 Conclusion

In this work, we presented a GUI retrieval system for Android applications that ranks GUIs from a semi-automatically created large-scale GUI repository based on NLR to facilitate GUI prototyping with customers. Our experimental results showed that standard IR models can be employed to effectively retrieve GUIs from NLR formulated as search queries or US. We also showed that AQE techniques could slightly improve the retrieval effectiveness of the BM25 base model.

Acknowledgements. This work is supported by the German Federal Ministry of Education and Research (BMBF).

References

1. Hiteshwar Kumar Azad and Akshay Deepak. Query expansion techniques for information retrieval: A survey. *Information Processing & Management*, 56(5):1698–1735, 2019.
2. Farnaz Behrang, Steven P Reiss, and Alessandro Orso. Guifetch: supporting app design and development through gui search. In *Proceedings of the 5th International Conference on Mobile Software Engineering and Systems*, pages 236–246, 2018.
3. Carlos Bernal-Cárdenas, Kevin Moran, Michele Tufano, Zichang Liu, Linyong Nan, Zhehan Shi, and Denys Poshyvanyk. Guigle: a gui search engine for android apps. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, pages 71–74. IEEE, 2019.
4. Daniel M Berry and Erik Kamsties. Ambiguity in requirements specification. In *Perspectives on software requirements*, pages 7–44. Springer, 2004.
5. Claudio Carpineto, Renato De Mori, Giovanni Romano, and Brigitte Bigi. An information-theoretic approach to automatic query expansion. *ACM Transactions on Information Systems (TOIS)*, 19(1):1–27, 2001.
6. F Dalpiaz. Requirements data sets (user stories). *Mendeley Data, v1*, 2018.
7. Biplab Deka, Zifeng Huang, Chad Franzen, Joshua Hibschman, Daniel Afergan, Yang Li, Jeffrey Nichols, and Ranjitha Kumar. Rico: A mobile app dataset for building data-driven design applications. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*, pages 845–854, 2017.
8. Biplab Deka, Zifeng Huang, and Ranjitha Kumar. Erica: Interaction mining mobile apps. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 767–776, 2016.
9. Lukas Galke, Ahmed Saleh, and Ansgar Scherp. Word embeddings for practical information retrieval. *INFORMATIK 2017*, 2017.
10. Forrest Huang, John F Canny, and Jeffrey Nichols. Swire: Sketch-based user interface retrieval. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–10, 2019.
11. Kristian Kolthoff. Automatic generation of graphical user interface prototypes from unrestricted natural language requirements. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1234–1237. IEEE, 2019.
12. Kristian Kolthoff, Christian Bartelt, and Simone Paolo Ponzetto. Gui2wire: Rapid wireframing with a mined and large-scale gui repository using natural language requirements. In *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20)*. ACM, 2020.
13. Christopher D Manning, Prabhakar Raghavan, and Hinrich Schütze. *Introduction to information retrieval*. Cambridge university press, 2008.
14. Kevin Moran, Carlos Bernal-Cárdenas, Michael Curcio, Richard Bonett, and Denys Poshyvanyk. Machine learning-based prototyping of graphical user interfaces for mobile apps. *arXiv preprint arXiv:1802.02312*, 2018.
15. Stephen E Robertson, Steve Walker, Susan Jones, Micheline M Hancock-Beaulieu, Mike Gatford, et al. Okapi at trec-3. *Nist Special Publication Sp*, 109:109, 1995.
16. Jim Rudd, Ken Stern, and Scott Isensee. Low vs. high-fidelity prototyping debate. *interactions*, 3(1):76–85, 1996.
17. Nakatani Shuyo. Language detection library for java. *Retrieved Jul*, 7:2016, 2010.
18. Didar Zowghi and Chad Coulin. Requirements elicitation: A survey of techniques, approaches, and tools. In *Engineering and managing software requirements*, pages 19–46. Springer, 2005.