CORDULA: Software Requirements Extraction Utilizing Chatbot as Communication Interface

Edwin Friesen
University of Paderborn
edwinf@mail.upb.de

Frederik S. Bäumer
University of Paderborn
fbaeumer@mail.upb.de

Michaela Geierhos
University of Paderborn
geierhos@mail.upb.de

Abstract

Natural language requirement descriptions are often unstructured, contradictory and incomplete and are therefore challenging for automatic processing. Although many of these deficits can be compensated by means of natural language processing, there still remain cases where interaction with end-users is necessary for clarification. In this vision paper, we present CORDULA, a system using chatbot technology to establish end-user communication in order to support the requirement elicitation and partial compensation of deficits in user requirements.

1 Introduction

User Requirements (UR) are one way to integrate end-users with a basic technical knowledge in the software development process. Instead of formally modeling the requirements of software, end-users write down which functions they expect from a software application. However, natural language allows incompleteness, vagueness and ambiguity in UR descriptions, which can significantly reduce the quality of the resulting software [GSB15]. While "ambiguity is the possibility of interpreting an expression in two or more distinct ways, [...] vagueness occurs when a phrase has a single meaning from a grammatical point of view, but still leaves room for varying interpretations" [GB17]. Although there are some (semi-)automated tools for the detection and compensation of deficits in UR [SJ15, Ban15, HB15], they cannot solve all deficits, due to the need of further context information.

The vision of On-The-Fly (OTF) computing[1] is to enable end-users to describe their software needs in natural language. Fitting these given UR, a software composing of microservices will be automatically generated [BG18]. Our current unidirectional compensation approach CORDULA (Compensation Of Requirements Descriptions Using Linguistic Analysis) [BG18] as the first application in OTF computing is able to detect and compensate deficits in UR by means of predefined strategies and indicators [BG18] (cf. example in Figure 1), but does not support the interaction with end-users. This is a procedure that is effective in cases of occurring lexical ambiguity and incompleteness in UR; but it is insufficient in cases of vagueness, for example. Vagueness cannot be compensated, because expressions like "*fast* applications" or "*large* files" provide no distinct information. We tried to work with standard configurations, but these exist in only a few cases. For this reason, these cases require user interaction in order to fulfill user's expectations of the resulting software. A similar, though not chat-based, approach can be found in RESI (Requirements Engineering Specification Improver) [Kör14], but the end-user is required to have a technical background. End-users get the chance to correct possible deficits within a dialog. With the chat we aim for higher assistance. The chatbot shall accompany end-users from start to finish while they build a software specification.

In the following sections we present currently existing challenges (Section 2) in CORDULA and how we plan to revise the system in order to solve the issues (Section 3). Therefore we give a concrete example of user interaction (Section 3.2). Finally we briefly conclude the benefits and expected new arising challenges (Section 4).

[1]For more information about OTF computing, visit `http://sfb901.upb.de`

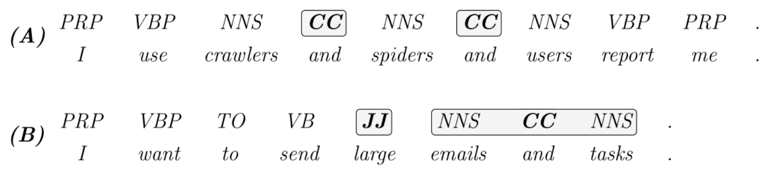

(A) PRP VBP NNS CC NNS CC NNS VBP PRP .
I use crawlers and spiders and users report me .
(B) PRP VBP TO VB JJ NNS CC NNS .
I want to send large emails and tasks .

Figure 1: Coordination ambiguity patterns (marked POS tags) as indicator for syntactical ambiguity [Bäu17]
A: [use crawlers] and spiders vs. use [crawlers and spiders]
B: large [emails and tasks] vs. [large emails] and tasks

2 Current Challenges

During the development and testing of the current version of CORDULA, we mainly faced four types of challenges.

The first two challenges are related to the compensation of incompleteness and inaccuracy in UR. There are always cases in which no gold standard exists for compensation (**missing information**) or in which several different suggestions have the same probability (**stalemate situation**). In such cases, the system cannot make a decision and relies on end-user interaction.

The third and fourth challenges arise from the different NLP components CORDULA uses. Heterogeneous results (e.g. POS tags) of individual components (e.g. requirement extraction and lexical disambiguation) can conflict each other (**contradictory information**). In the past, we tried to solve this problem by assigning some priority to one of the conflicting components, which means that one partial result is overwritten by another. However, the predominant component may have been wrong in its decision and destroys the overall result. For example, if the requirement extraction has recognized a process word ("to display") and the component for lexical disambiguation detects this word as a noun ("display"), it will overwrite the previous POS tag. As a result, no structured software requirement can be generated, because the core process word (Action) is missing and the requirement will simply be ignored. It would be easy to detect this error because there has to be at least one process word per sentence, but the final decision must be made by the end-user in this case (**known but not solvable problems**).

Sometimes situations arise in which information must be provided subsequently or the system is unable to cope with a given input. However, it cannot be assumed that end-users will be able to understand the identified problems at once and solve them. For this reason, we develop a Knowledge Base (KB) and enrich it with instructions and examples. Moreover, it is not sufficient to attach a chat functionality to the existing system, but it must become the single point of contact for the users. For this reason, several design questions arise in addition to conceptual questions.

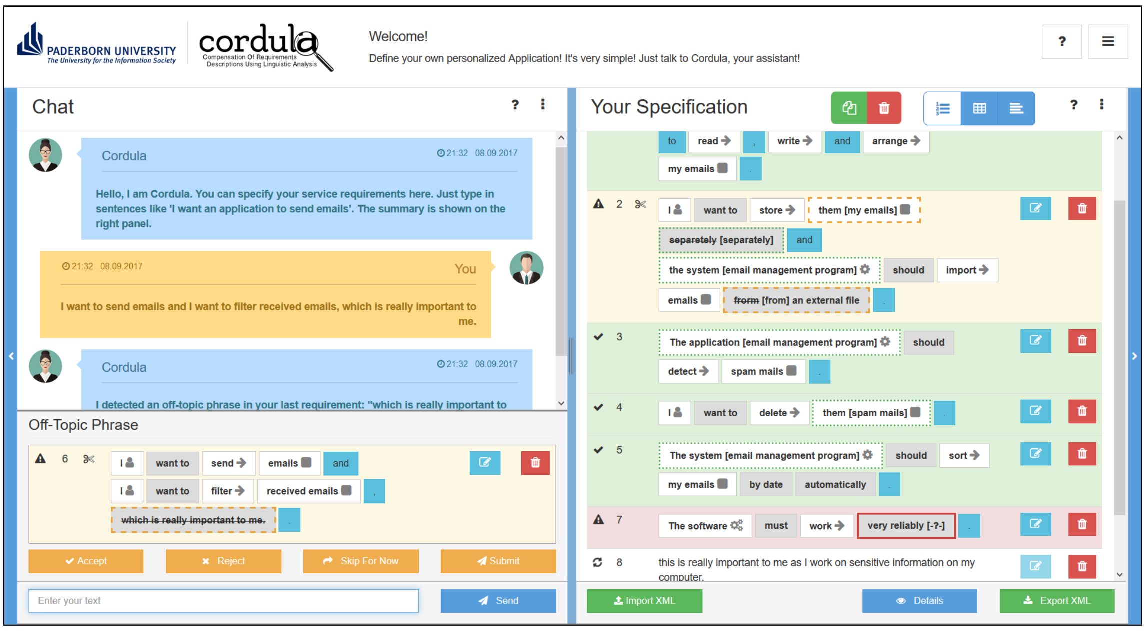
3 Vision: Let's Talk with CORDULA

The revised version of CORDULA aims to overcome the named weaknesses of the existing system. This already begins with the design of the web interface, which has to be changed in that a dialog between end-users and the system is in the centre of attention (cf. Figure 2) and also affects fundamental system components such as the KB or the internal communication between the system components.

The idea of using a chatbot has a strong effect on the underlying system architecture of the current version of CORDULA. Until now, the entire processing pipeline was concentrated on a static input text. Now, new information can be added which can also affect already processed requirements. For this reason, a requirements manager should be installed to monitor the effects of the user dialog as well as the changes in the software requirements made via the GUI.

3.1 Interactive GUI

The main layer shown in Figure 2 is divided into two parts: a chat window on the left and the "specification box" on the right. The chat window expects input from end-users in order to provide requirements and additional information. The user input will be forwarded to the server and processed. The server's response will be articulated via the chatbot. Inaccuracies are pointed out and a selection of tailored action proposals is offered (e.g. context-specific suggestions to compensate incompleteness, cf. Figure 3). While chatting, the end-users can confirm the system's suggestion or reject it. Additionally, the requirement can be edited or deleted. This means that users communicate with the system in two ways. On the one hand, users transfer the requirements



Welcome!
Define your own personalized Application! It's very simple! Just talk to Cordula, your assistant!

Cordula 21:32 08.09.2017
Hello, I am Cordula. You can specify your service requirements here. Just type in sentences like 'I want an application to send emails'. The summary is shown on the right panel.

21:32 08.09.2017 You
I want to send emails and I want to filter received emails, which is really important to me.

Cordula 21:32 08.09.2017
I detected an off-topic phrase in your last requirement: "which is really important to

6 I want to send emails and
I want to filter received emails ,
~~which is really important to me.~~ .

to read , write and arrange
my emails .

2 I want to store them [my emails]
~~separetely~~ [separately] and
the system [email management program] should import
emails ~~frorm~~ [from] an external file .

3 The application [email management program] should
detect spam mails .

4 I want to delete them [spam mails] .

5 The system [email management program] should sort
my emails by date automatically .

7 The software must work very reliably [-?-] .

8 this is really important to me as I work on sensitive information on my computer.

Figure 2: Mock-up of CORDULA's new GUI (left: chat and interaction window; right: current specification)

to CORDULA via the chat interface and receive an immediate response. On the other hand, they will be asked to react to certain circumstances (e.g. missing information) and get a chat that adapts to the situation. These interaction possibilities are directly related to the chat history (cf. Subsection 3.2), in which further instructions and examples can be given by CORDULA. As shown on the right-hand side of Figure 2, end-users can oversee and edit the software requirements. This view also contains additional information on semantics and (lexical) annotations and is considered as a debugging view. In order not to discourage the end-users, CORDULA should briefly explain the functions at the beginning of the conversation. End-users should continue only transmitting their requirements and, in the best possible case, not take any further action at all.

3.2 Sample Dialog

For a better understanding of CORDULA's functionality, we illustrate an exemplary conversation and interaction of CORDULA and a user below. The dialog below (Table 1) is a protocol of how the actors and system components could interact, demonstrated through a vague user requirement.

Actor	Action	Notes	Reference
User	<accesses CORDULA's web GUI>		Figure 2
Bot	<explains functionality, offers help>		
	<asks for requirements, gives examples>		
User	"I want to send large emails."		
System	<NLP>	structures, annotations etc.	
REaCT[2]	<Requirement Classification: true>	input: Chat or Requirement?	[DG16]
	<OnTopic Classification: true>	input relevant/on topic?	[DG16]
	<Semantic Information Labeling: Role, Priority, Action, Object>	requirement labels for further analysis	Fig. 3, [DG16]
System	<Indicator Checking>	pattern/rule-based search	Fig. 1, [Bäu17]
	<Token "large" classified as vague>	e.g. list matching or from KB	
	<Action: send , Object: emails $\rightarrow$ Email-Service added>	KB: *Action-Object* relation corresponds to Service	
Bot	"A vague term has been found: *large emails.*		
	Please use concrete values (e.g. *500 MBs).*"		
User	<clicks the edit button>		Figure 3

[2]Requirement Extraction and Classification Tool [DG16]

	<"I want to send large emails *of at least 200MBs.*">		
	<submits the changes>		
System	<processing like before>		
	<requirement and specification is updated>		
Bot	"Your requirement seems to be correct now. New functionality *E-Mail Service* added to your specification."	templates used by bot for prompts and responses	

Table 1: Possible dialog flow in CORDULA

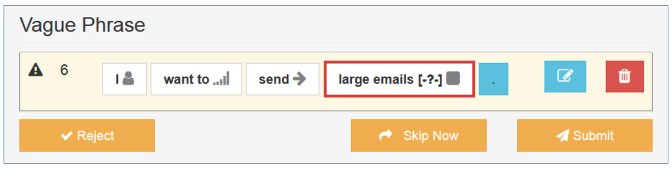


Figure 3: Interaction takes place in the interaction box below the chat (cf. Figure 2) containing the latest requirement. The red frame indicates an issue, the question mark stands for an unsolvable problem (cf. Section 2). The icons represent the semantic information labels: namely *Role*, *Priority*, *Action* and *Object*.

4 Conclusion & Research Outlook

As we have shown, our current approach for the automatic detection and compensation of linguistic inaccuracies and incompleteness is not able to solve all deficits (cf. Section 2). Therefore, we still depend in part on information provided by end-users. However, since these users have no technical background, they must be supported by a chatbot to be efficiently guided through this process (cf. Section 3). At this point, as we have pointed out, it is not sufficient to simply add a chatbot to the previous system. Rather, the entire architecture of the system should switch to interaction and also adapt the design of the GUI. Here, we already outlined some changes we expect to see. However, we expect further challenges, which will accompany us in the development of the next version of CORDULA. The vision described here is only a beginning.

Communication: In the current version of CORDULA, end-users transfer a coherent text to the system as one RD. This puts the individual statements in context. A chat dialog, as we want to introduce it, breaks up this context, which can make processing, such as disambiguation, more difficult. Furthermore, the text quality could further decrease because end-users use a kind of "chat language". Both cases make it necessary to revise the current requirement extraction and classification and to train for the new text genre.

Knowledge base: NLP resources in the area of UR are very rare [TLK15]. This includes resources for detection and compensation as well as real UR, which are needed for machine learning approaches. This problem is aggravated by the lack of instruction texts and examples as needed in our chat approach. Users are supported by the system only if it is very close to their own requirement description in domain and wording. A rigid collection of example sentences does not do justice to this claim. An approach for context-sensitive help and requirement examples must be found here.

We believe that we meet the requirements of OTF computing for a fast execution, even with a slower chat solution. The composition of services is a complex task whose success also depends on correctly recognized UR.

Acknowledgements

This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901).

References

[Ban15] Muneera Bano. Addressing the Challenges of Requirements Ambiguity: A Review of Empirical Literature. In *Proceedings of the 5th International Workshop on EmpiRE*, pages 21–24, Ottawa, ON, Canada, August 2015. IEEE.

[Bäu17] Frederik Simon Bäumer. *Indikatorbasierte Erkennung und Kompensation von ungenauen und unvollständig beschriebenen Softwareanforderungen.* Phd thesis, University of Paderborn, Paderborn, Germany, July 2017. ISBN: 978-3-942647-91-5.

[BG18] Frederik S. Bäumer and Michaela Geierhos. Flexible Ambiguity Resolution and Incompleteness Detection in Requirements Descriptions via an Indicator-based Configuration of Text Analysis Pipelines. In *Proceedings of the 51st Hawaii International Conference on System Sciences*, pages 5746 – 5755, Big Island, Waikoloa Village, USA, 2018.

[DG16] Markus Dollmann and Michaela Geierhos. On- and Off-Topic Classification and Semantic Annotation of User-Generated Software Requirements. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1807–1816, Austin, TX, USA, 2016.

[GB17] Michaela Geierhos and Frederik Simon Baeumer. Guesswork? Resolving Vagueness in User-Generated Software Requirements. In Henning Christiansen, M. Dolores Jimenez Lopez, Roussanka Loukanova, and Lawrence S. Moss, editors, *Partiality and Underspecification in Information, Languages, and Knowledge*, Partiality and Underspecification in Information, Languages, and Knowledge, chapter 3, pages 65–108. Cambridge Scholars Publishing, September 2017.

[GSB15] Michaela Geierhos, Sabine Schulze, and Frederik Simon Bäumer. What did you mean? Facing the Challenges of User-generated Software Requirements. In *Proceedings of the 7th International Conference on Agents and Artificial Intelligence*, pages 277–283, 2015.

[HB15] Shahid Husain and Rizwan Beg. Advances in Ambiguity less NL SRS: A review. In *Proceedings of ICETECH 2015*, pages 221–225, Coimbatore, TN, India, March 2015. IEEE.

[Kör14] Sven J. Körner. *RECAA – Werkzeugunterstützung in der Anforderungserhebung.* PhD thesis, Karlsruher Institut für Technologie (KIT), Karlsruhe, Germany, February 2014.

[SJ15] Unnati S. Shah and Devesh C. Jinwala. Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey. *SIGSOFT Software Engineering Notes*, 40(5):1–7, September 2015.

[TLK15] Walter F. Tichy, Mathias Landhäußer, and Sven J. Körner. nlrpBENCH: A Benchmark for Natural Language Requirements Processing. In *Multikonferenz Software Engineering & Management 2015*, pages 159–164, March 2015.