computers

MDPI

Article

Natural Language Processing in OTF Computing: Challenges and the Need for Interactive Approaches

Frederik S. Bäumer *, Joschka Kersting and Michaela Geierhos

Semantic Information Processing Group, Paderborn University, 33100 Paderborn, Germany; jkers@mail.upb.de (J.K.); Geierhos@mail.upb.de (M.G.)

* Correspondence: fbaeumer@mail.upb.de; Tel.: +49-5251-60-5666

Received: 22 January 2019; Accepted: 3 March 2019; Published: 6 March 2019

Abstract: The vision of On-the-Fly (OTF) Computing is to compose and provide software services ad hoc, based on requirement descriptions in natural language. Since non-technical users write their software requirements themselves and in unrestricted natural language, deficits occur such as inaccuracy and incompleteness. These deficits are usually met by natural language processing methods, which have to face special challenges in OTF Computing because maximum automation is the goal. In this paper, we present current automatic approaches for solving inaccuracies and incompletenesses in natural language requirement descriptions and elaborate open challenges. In particular, we will discuss the necessity of domain-specific resources and show why, despite far-reaching automation, an intelligent and guided integration of end users into the compensation process is required. In this context, we present our idea of a chat bot that integrates users into the compensation process depending on the given circumstances.

Keywords: inaccuracy detection; natural language software requirements; chat bot

1. Introduction

Software requirements are challenging from a user perspective because they often allow a high degree of freedom in project implementation due to inaccuracies [1]. The idea of On-The-Fly (OTF) Computing (Additional information about OTF Computing; https://sfb901.upb.de).

employs individual software requirements in natural language (NL) provided by users for an automatic composition of individual software services. Here, it is challenging that the requirements are provided in NL and are thus partially incomplete, inconsistent or ambiguous. Though there are various applications designed to deal with NL issues of this kind, these are imperfect and have weaknesses. A concrete example therefore is the compensation of missing non-common information with default or heuristic values [2]. Since this procedure obviously is not accurate and individual enough, especially in the context of OTF Computing solutions, users may be dissatisfied with this manner. Since the bidirectional dialog between software developers and end users is omitted in the OTF Computing vision, new ways are needed to obtain missing information from end users regarding the desired software, which currently cannot be found in any existing linguistic resources [2]. Research as well as practical tools for NL requirement refinement are often dedicated to special domains or designed with different guidelines that do not require a fast computation or other OTF-typical standards. However, most approaches are developed for experts rather than for end users or an application in everyday life. Consequently, we want to draw attention to challenges of service description processing in the context of OTF Computing, where a very high degree of automation is required, but communication with end users is sometimes necessary, for example when information is missing. We show the resulting implications for Natural Language Processing (NLP) and further research in this context in Section 2. The current state-of-the-art is presented in Section 3, while discussion takes place in Section 4.

On the basis of this, we present our current approach for domain-specific knowledge representation in Section 5.1, before we describe our work on dialog-driven compensation systems (Section 5.2). Finally, we provide a conclusion and plans on our future work in Section 6.

2. Service Descriptions for OTF Computing: Open Challenges

OTF Computing aims at providing tailored software services to individuals. Here, NLP is crucial for the functionality of the OTF vision. Thus, being the only possible form for service descriptions, we have to tackle NL shortcomings while finding and making use of as much information as possible from given NL input. More formally described, service descriptions are a less formal subtype of requirement specifications because they describe a service in NL. Nevertheless, the individual factors such as the user's knowledge or some expert level know-how can shape a requirement and its accuracy as input for other processes. This leads to service descriptions being characterized as user-generated, informal documents [3]. However, when it comes to OTF Computing, formal or semi-formal description languages are a method of choice for tackling NL shortcomings. Here, the Software Specification Language is an example for a language that aims at a comprehensive specification of services while being applicable to non-functional and functional software requirements [4]. Nevertheless, formal specification languages, even in a weaker form, are not applicable for end users [5] because they are neither experts nor have the corresponding knowledge [6]. After all, formal and semi-formal approaches bring invincible issues with them, which force system operators to tolerate freely formulated NL requirement descriptions by end users. They have to face typical NL challenges, among which are a lack of structure and correctness, some grammar and spelling errors as well as occasional ambiguity issues. This is still a tremendous difference compared to the usefulness of formal specification methods already available. We want to focus on the following three issues in this paper (Sections 2.1–2.3).

2.1. Extraction of Canonical Core Functionalities

It is a challenging task to find important and useful information in NL service descriptions [7], as they are unstructured and contain off-topic information [5]. Apart from that, canonical core functionalities are neither provided in an order, nor easy to extract. Thus, functional requirements can be described as semantic categories such as roles, objects, etc. [7]. For further steps, relations among requirements are important in order to figure out which requirements determine others. These steps specify which function an application might have or which services are booked. This requires extracting all specified functionalities and ordering them functionally. Our example "I want to send e-mails to my friends: First, I need to write them and then I want to attach my files" can be found in Figure 1. The state-of-the-art puts the functions in a temporal or causal order, a way the user might have intended.

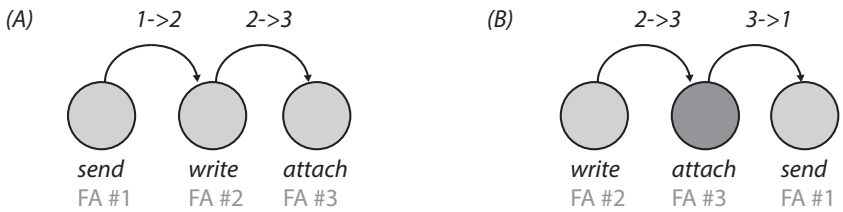

Figure 1. Comparison of generated functional sequences [8] (**A**: Given seq.; **B**: Temporal seq.).

However, there are issues with this procedure as well. As described in Figure 1A, it cannot be executed because an e-mail cannot be sent before it is written and enriched with attachments. Thus, the temporal sequence is relevant to process words. Figure 1B supports this. Considering Figure 2, the hierarchy of requirements is also important to subject words. That is, to "send" is primarily a process word that presupposes some e-mail text and an attachment before the delivery can take

place. Finding a sequence of process words is not enough. The concepts mentioned have to be extended consequently in order to find all required steps and thus functions. An example is "attach", which requires steps taking place before attaching a file. At first, this file has to be chosen ("choose"). This term was not explicitly mentioned in the service description in Figure 2. To tackle such cases, we need some knowledge base that is capable of dealing with this. Such a database has to contain process words and their hierarchy, i.e., semantic relationships, all dependencies and further relations such as synonyms and further domain information. However, such a resource does not exist today. Furthermore, there exists no method to find interrelationships among requirement descriptions in a detailed manner as it is required in our scenario.

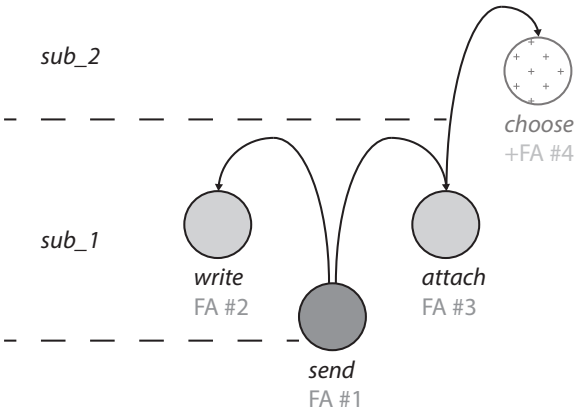

Figure 2. Hierarchical arrangement of process words [8].

2.2. Automatic Detection and Compensation of Inaccuracy

Inaccuracies such as vagueness and ambiguity are well investigated topics (cf. e.g., [5,9–11]). This is an especially important research field to OTF Computing, in which service descriptions are given in NL. Here, issues such as ambiguity or inaccuracy must be expected due to the lack of expert knowledge on the end users' side. However, software oftentimes is designed or composed according to requirement descriptions but does not meet the intended use cases or general requirements of stakeholders [12] and the result can lead to serious errors [13]. Of course, as in many other cases, inaccuracies can be eradicated with a manual detection e.g., by making use of checklists [14] or with the support of semi-automatic applications [8]. We focus on software that, on the one hand, detects various types of inaccuracies and, on the other hand, identifies linguistic characteristics leading to inaccuracy. Software has to be able to solve as many deficits as possible in the NL descriptions independently and yet in case of doubt be able to automatically communicate with end users to solve the problem. We here need a kind of knowledge base in the form of linguistic resources as a minimum, which are rare for the domain of software requirements [15]. There is even less in the OTF Computing domain [16]. The disambiguation service Babelfy is often used as a substitute for a domain-specific disambiguation but is hosted entirely on external servers [17,18]. Despite good performance [19], server failures, network errors and response times would be a threat when integrating this service in a live OTF Computing system [8]. In addition, the integration of third-party software oftentimes does not work, due to a lack of programming interfaces, but is crucial for an automatic and quick system. All of this means that resources and domain-specific compensation procedures have to be developed for the OTF Computing vision.

2.3. Explainable Results

As there are various ways to elaborate software requirements, the question of explainability comes up when many improvements have been made. That is, reasons for correcting specific spans in the requirements and reasons for the improvements need to be comprehensive for users. Thus,

end users have to be informed in order to understand the service composition and be satisfied with its characteristics. So far, corrections are in part highlighted or compared to the input (e.g., [19]); however, there is no explanation information for the changes. There are several challenges in the OTF context: the NL input of end users mainly shapes the software composition even though the type of composition (the actual software service selection step) has an influence as well. This results in a transparent elaboration process that covers the methods for detection and compensation of NL inaccuracies to enhance the end users' understanding.

3. State-of-the-Art

This section deals with the current state of research in the domain of canonical core functionality extraction as well as automatic inaccuracy compensation and detection.

3.1. Requirements Extraction

Up to now, there is little research done on extracting software requirements. For example, there is a tool named REaCT [7,20], which makes use of learning procedures to find phrases belonging to a certain domain or topic. It works on textual requirement descriptions and tries to transfer the most important entities for functional requirements found in a template that can be used further. From the technical side, Ref. [20], this involves dividing description texts into sentences and classifying them into off- and on-topic components. After this step, from the on-topic sentences containing functional requirements, there are attribute-value pairs extracted in order to iteratively fill the template, starting with the most important elements such as subjects, actions, predicates and objects, e.g., indirect objects. However, even though this approach convinces with good performance values, there are no suitable resources to be used and it therefore fails [16]. Here, textual software requirements can be helpful though not or rarely being available [15]. This leads to the idea of extracting requirements without machine learning but rule-based. At any rate, due to the possibly low quality of descriptions, there is still enough to be done. Other approaches focus on high-quality requirements or assume receiving high-quality texts, which makes such tools unsuitable for the OTF Computing domain. Apart from that, there is a study of unstructured and informal requirement descriptions from the Open Source domain [21].

3.2. Multiple Inaccuracy Detection and Compensation

In literature, the idea of combining different approaches for inaccuracy detection or compensation was investigated. Ambiguity and incompleteness together were covered (e.g., [22–24]) as well as ambiguity only (e.g., [25,26]). Furthermore, there are researchers giving an overview over disambiguation methods in the domain of NL software requirements [27,28]. The automation degree, chosen methods (rule-based, ontology-based, etc.) and technologies used (e.g., Stanford Parser) are categories drawn by Shah and Jinwala [28]. Another researcher provides an overview for empirical work [29].

Here, we focus on combined approaches for the detection and compensation of ambiguity and incompleteness. QuARS [30] and QuARS$_{express}$ [31] can deal with a broad range of inaccuracies while NL2OCL and SR-Elicitor are tools that reach a low coverage. Moreover, the approaches have different aims. QuARS is supposed to detect a high number of issues in requirement descriptions, while NL2OCL [26] and SR-Elicitor [10] should detect and compensate issues fully automatically. Another tool, RESI (Requirements Engineering Specification Improver), is based on a high degree of user interaction when compensating inaccuracies. We here want to draw attention to three solutions as we describe them in more detail for a better understanding. These tools are NLARE (Natural Language Automatic Requirement Evaluator) [23,32], RESI [33] and CORDULA (Compensation of Requirements Descriptions Using Linguistic Analysis) [19]. Furthermore, we highlight the discrepancies here.

NLARE is a hybrid approach with focuses on functional requirements and the detection of ambiguity, incompleteness and atomicity [23]. Among other things, the software employs an atomicity

criterion that basically sets the rule that a single sentence must contain a single requirement. Apart from that, incompleteness is seen as to complement information dealing with "W-questions": "Who", "What", "Where", "When". The authors regard ambiguity as given when adverbs and adjectives occur that can be de- or increased. NLARE makes use of regular expressions in order to process NL data. NLARE further applies spelling correction, detects sentence boundaries and tokenizes words. The users of this tool get simple hints such as "The requirement is ambiguous because it contains the word 'earlier' and 'later"' as a result [23]. There is no assistance or compensation of inaccuracies. However, there are other tools: RESI [33] has a different aim because it enables (and encourages) user interaction while being flexible and also dealing with linguistic defects.

RESI understands requirement specifications as a graph where it automatically identifies inaccuracies. Each inaccuracy is solved within a dialog between the user and the system [33]. Here, RESI goes beyond indicating inaccuracies by providing assistance for solving them in various cases (e.g., incomplete process words). Thus, when integrating large resources, deficits can be found and solved using the additional knowledge. This additional information might covered by different ontologies or comparable information sources. Especially with the further development of the Semantic Web [34] and Linked (Open) Data approaches [35,36], additional possibilities arise to integrate increasingly structured knowledge from different domains. Much knowledge in the field of requirements engineering and NLP is already available online but cannot be used automatically [16].

Moreover, there is a tool called CORDULA, which is in some cases similar to RESI [19]. CORDULA is able to find and compensate "language deficiencies" such as ambiguity, vagueness and incompleteness in written requirement texts produced by end users. Thus, CORDULA is well suited for the OTF Computing domain, for which it was also designed and developed. CORDULA further enables users to find suitable software and services that can be used to generate canonical core functionalities from service and requirement description texts. Predefined linguistic features, used as indicators, enable the system to improve text quality individually. This approach is data-driven and aims at current needs as it is driven by a typical text analysis pipeline. A core feature that distinguishes CORDULA from other systems is the ad hoc configuration of the compensation modules in the pipeline. This is also determined by deficiencies that were identified in the service specifications written by users. However, CORDULA has a considerable disadvantage for the OTF Computing domain: its slow execution time [8]. This leads to drawbacks: i.e., no compensation can be undertaken when there is contradictory information from several compensation methods. Furthermore, when information is just missing, there cannot be any compensation performed [19]. There exist various tools that can detect different types of inaccuracies in NL service and requirement descriptions. Some of them can even compensate issues. Nevertheless, none of these tools meet all premises for NL service descriptions in the domain of OTF Computing.

4. Open Challenges: A Discussion

In requirements engineering, NLP and corresponding pipelines were of great interest in the research community for a long time. Usually, NLP is used to analyze the textual service descriptions produced by end users in order to enhance them while maintaining the original intention of the corresponding text. Still, inaccuracies should not lead to ordinary users being bothered with technological or linguistic details. This is a crucial as well as challenging task. There is a prototype trying exactly this—the NL pipeline creates structured compositional templates from unstructured service descriptions [8,19]. Still, there are many unsolved questions, especially as several smaller objectives such as a high performance have not been accomplished. Most existing tools were developed for specific linguistic phenomenons serving only for special cases. Nevertheless, approaches that follow a meta-strategy are required while keeping track of individual issues in each of the requirement specification texts. It is important to find out whether solving vagueness affects ambiguity. Moreover, it has neither been questioned nor answered whether it is necessary to compensate a requirement in order to fully grasp the requirement's meaning—e.g., does the ambiguity concern essential elements

of a requirement, or can the compensation be skipped in terms of performance? Apart from that, the synergy between various compensation components is not investigated. Here, new insights might help building an almost optimal system by making the optimal use of every component. As an example, information from the semantic disambiguation might help improving the processing time of another step. Thus, the single components of a NLP pipeline should flexibly interact with each other for interchanging information as much as possible. This dissolves the common procedure of a pipeline as being a sequential technology. Since end users will not regard every requirement necessary for a fully working service composition, it is not sufficient to extract requirements from textual descriptions and present them in a structured way. Furthermore, users write their descriptions on different abstraction levels. Of course, on the non-expert level, they know from software usage when they "want to send e-mails", for example. The backend level of the components is non-existent to most ordinary users. Clearly, the parts for extracting and compensating information must work well with each other in order to detect hierarchical dependencies between process words and find links among them.

Nevertheless, domain-specific resources that cover specific properties are missing, as mentioned. However, there indeed exist resources such as BabelNet (Visit https://www.babelnet.org for more information.) (cf. Section 5.1). However, apart from linguistic information, domain-specific information needs to also be added, in order to cover the domain in wording and knowledge. The hierarchical information between process words can serve as an example. Figure 3 presents an exemplary knowledge base the way we would construct it from a rather theoretical point of view. Besides linguistic modeling (such as BabelNet), we include processing relationships such as "action" and "object", which represent semantic information about the dependencies between information from the perspective of requirements engineering. At any rate, apart from a rather theoretical starting point, it is basically the same approach to start data-driven and model such relationships for existing data in order to have not only theoretical but real-life samples that, enriched with linguistic and requirements information, can be used for semi-automatic and automatic requirement engineering.

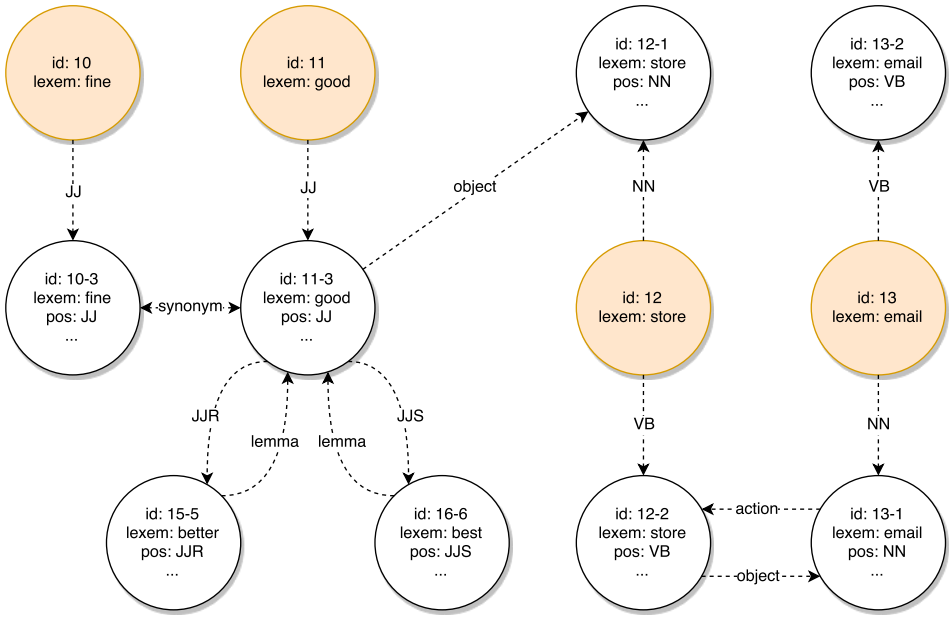

Figure 3. Theoretical concept of a NL requirement knowledge base.

However, there are no linguistic resources that cover characteristics of requirement specification texts collected in the OTF Computing domain [15,16]. That is, there are neither resources for compensation, nor for providing examples with real service descriptions. Indeed, this is challenging as we can expect only plain textual NL descriptions from users. However, and this is a vicious circle, the system works only if we have such texts for training and testing. However, there is an approach using

texts that are at least related to our scenario [7], although service descriptions are still a bit different. Still, several features are not covered or different [16].

Next to the discussion of adequate resources, there is the discussion of whether and how users might be involved in the compensation process. This is a constant point and especially important according to the requirements an OTF Computing system has. In most cases, the strategy was that the more automation is achieved, the better. However, users here are excluded from the compensation process even if they might be qualified and willing to assist. This might also happen due to the fact that user collaboration is time consuming. Since users know their own needs and desires (i.e., requirements) best, interaction seems reasonable. Of course, no one but the users themselves can say what they as individuals actually expect. Users can quickly dissolve ambiguity or make other useful decisions if the system supports them [37]. A chat bot could support users in a dialog. Here, users can be integrated in the process of requirements analysis and refinement. For example, the chat bot can provide examples ("Did you mean X like in this sentence: "..." ?"), or any other help that fits to a certain situation [37]. Furthermore, inaccuracies can be highlighted by the chat bot. The users can confirm suggestions made by the bot or reject it. In short, users can use two channels in the chat: first, they can describe their requirements in short texts or sentences. They receive a response for this immediately. Second, they are urged to react to proposals provided by the chat bot. This includes missing information, among other things, and the bot can adapt to the situation. However, it quickly becomes apparent in this context that dialog-controlled compensation processes cannot be implemented with existing chat bot frameworks for everyday communication, because domain-specific resources are also necessary here. Thus, it is not a question of asking for the weather or starting a certain action on a smart phone, but of answering complex questions in requirements engineering together with end users and supporting them in the best possible way with examples and hints. We therefore present our work in Section 5.2.

5. Domain-Specific Approaches

In the following, we present two research directions that we follow to solve the open challenges mentioned above. On the one hand, we present our work on a knowledge base, which can be used for domain-specific NLP components as well as for end user interaction. On the other hand, we present our chat bot concept, which allows end users to easily interact with the NL compensation system.

5.1. Knowledge Base

As previously stated, there are no adequate linguistic resources that can be used for providing knowledge in the OTF Computing context. A chat bot without any background knowledge cannot interact accordingly. Hence, we need to built our own knowledge base. The current work on it will be presented in this section.

5.1.1. Motivation: A Domain-Specific Knowledge Base for the OTF Computing Scenario

At first, to establish an understanding for the need for such a data base in a practical sense, we provide a simple example case: a user wants to send e-mails and consequently provides the following input to the chat bot: "I want to write mails to my friend John." In the end, the system needs to (1) know (i.e., find out) that the user wants to write a mail; (2) send it, which is not explicitly mentioned in the text; (3) know that John is a person and what his mail address is; (4) process possible further requirements such as that the system needs access to a contact database or previous mails as the user could later mention to not initially write a mail but to answer a previously received mail to the friend. As can be seen, a multitude of information is covered here. However, when using a knowledge base, the system can deal with this by providing examples such as: Do you want something similar as in the sentence "Users are able to write mails, attach files and send them to their friends stored right in the app's own contact data base"? Of course, several other use cases are possible. However, there are also dozens of challenges on the way to create a knowledge base that "minimizes entry barriers and fosters a softer adoption" [35], such as those discussed in the research area of Linked Data [38].

5.1.2. Development of a Linked Knowledge Base

In order to build a knowledge base, we chose the practical, data-driven way: that is, we acquired a data set of requirement description-related texts and further processed them to have semantic and syntactic knowledge as well as domain-dependent information. The result can be imagined as a database full of examples that went through numerous processing steps. In the following, we describe the process of building the knowledge base and present statistics.

Our database was filled with software description texts collected from Download.com (See https://download.com, for further information.), which represents a type of app store. The texts used there come close to software requirements. There are oftentimes descriptions of what the corresponding application is capable of. This data has been used before [8] and is comparable to open source descriptions that were used as well in the same domain [7]. All collected texts were preprocessed: those being too short, in the wrong language or typeset, etc. were excluded from further processing, so roughly 217,000 texts remain. These were split into 1,050,359 sentences, of which several were excluded because of being too short or long and thus not helpful for any potential application. Here, on the sentence level, we started with the in-depth analysis. However, most importantly, we analyzed the sentence structure and the relations among tokens as well as characteristics of the tokens. There are 19,167,224 tokens in all sentences that are based on 281,002 lemmas. Here, tokens are unique, because their properties and relations to other tokens depend on the corresponding sentence and other tokens in it. In general, texts are connected with sentences appearing in them; these are connected with their tokens which have connections to other tokens, lemmas and, most importantly, to their senses. The senses are derived from the previously mentioned BabelNet, which offers a disambiguation service called Babelfy. Even if our solution depends on an external resource where further information about a sense can be found, this was a sufficient way of dealing with ambiguity. There are 143,918 senses and about 92 million relationships between all nodes (i.e., texts, sentences, tokens, . . .). Above all, there are 20,859,748 nodes. The vast number of nodes and relations indicates a high complexity of the knowledge base. An example can be found in Figure 4. Here, one text (yellow) is connected to its sentence (green), the relations are typically named “hasSentence”, “hasSense”, etc. The tokens (blue) of one sentence are displayed as well. As can be seen, they are all connected to a sentence (“hasToken”) and to each other. Furthermore, we display the lemma and sense of the token “send”. The lemma (purple) is also “send”. The lemma “send” is connected to any other token that is a form of send, regardless of its meaning, in every other sentence in the knowledge base. The sense here (red) includes a “babelID”, a unique resource identifier from BabelNet which we keep to enable the allocation of our resource in the Linked Data area. In this field, we will introduce and adopt further vocabulary to enable maximum compatibility in the Semantic Web. Nodes of type tokens are enriched with various information. However, Figure 4 is a view that does not include every relation of the nodes. Furthermore, for simplification, properties are not shown here.

As we plan to use our knowledge base within a chat bot where user input is rather short, the main analysis was undertaken on a sentence basis, emphasizing on the tokens. This is implied by Figure 4, as most relationships exist between tokens. However, the main processing steps were dependency parsing and semantic role labelling (SRL) [39,40]. These two steps help analyze the sentence structure. For example, using SRL, arguments in a sentence can be found. If the chat bot receives a requirement specification that misses the second argument in a sentence, we can provide an example sentence to make the user complete his/her initial specification. Thus, the example sentence can inherit the same verb with the same meaning and all required arguments: “You want to write a mail. Do you want to do something like this sentence (?): I want to write and send mails to my contacts.”. Furthermore, we included named entity recognition, token shapes (shape of “House” is e.g., “Xxxxx”), detailed part-of-speech (POS) tags, position in the sentence and a unique ID that links the token to its sentence. Furthermore, all nodes and relationships have a binary property that states whether its information was checked manually.

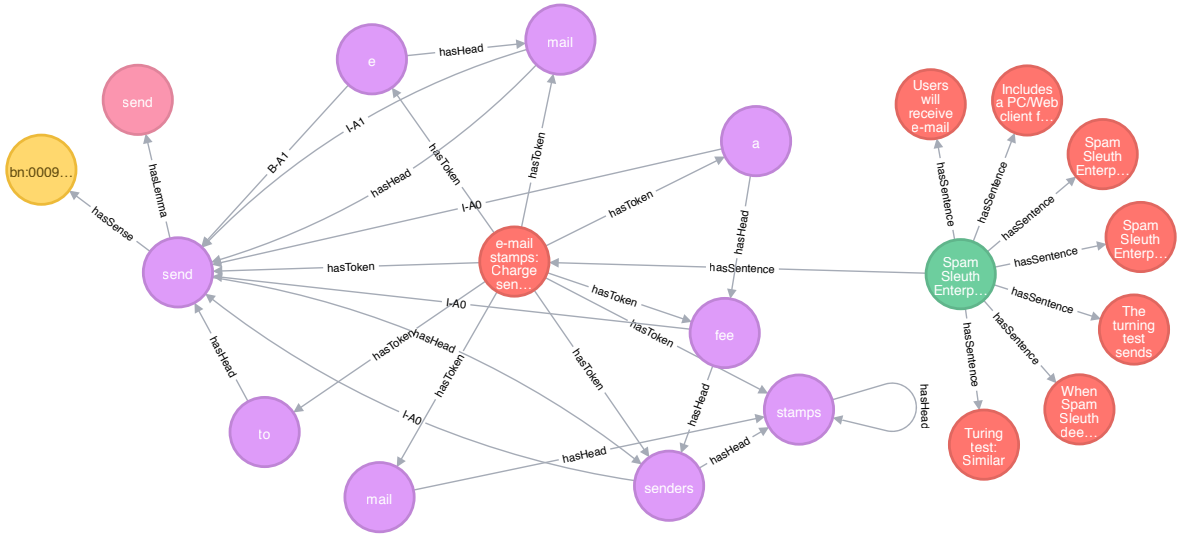

Figure 4. Graph sample of an NL requirement knowledge base.

In general, our knowledge base is a large resource that provides us millions of software descriptions that were linguistically and semantically analyzed. Moreover, several analyses such as tagging and shapes were added. In the following, this resource can be used to quickly present examples to a user in a chat bot dialog or to compare the input to example sentences. Additional enrichments such as relations like "IsPartOfService" are not yet included but are possible to add later ("send" and "mail" are part of "e-mail service"). In principle, the next step of our work shall be to strive for an alignment with existing vocabulary on the Semantic Web in order to achieve a high level of reusability and extensibility and to make the concepts used applicable beyond the OTF domain [38].

5.2. Inaccuracy Compensation by Using Domain-Specific Chat Bots

For the implementation of domain-specific chat bots, it is essential to carry out interactive composition processes in a direct dialog. This is essential for specifying and completing existing requirement descriptions, especially for iterative clarification processes. Ultimately, these are dialogs between people and computers. Unless a strict question-answer-scheme is used, we generally speak of NL dialogs. Their computational interpretation requires a robust and flexible dialog control, in particular due to a highly individual chat flow.

5.2.1. Concept of a Chat Bot with Dialog Control

To eliminate requirement deficits that cannot be compensated by existing methods together with end users, dialogs are necessary that contribute to the specification and completion of derived requirement specifications. Existing extraction and compensation methods create preliminary service templates on the basis of which software services can be selected and composed. The goal of dialog control is to iteratively revise deficient templates in a guided dialog in order to be able to provide the necessary information on the service composition as far as possible. To achieve this, we subdivide the dialog control into a requirement interpretation and a chat interpretation (cf. Figure 5). While the latter is responsible for the dialog guidance and interpretation of all user inputs, the requirement interpretation is responsible for the classification, interpretation and validation of recognized requirements. This separation makes it possible to integrate existing chat bot techniques (established dialog guidance), while the processing of software requirements can fall back to the extraction and compensation components developed in earlier work (CORDULA, cf. Section 3) [19].

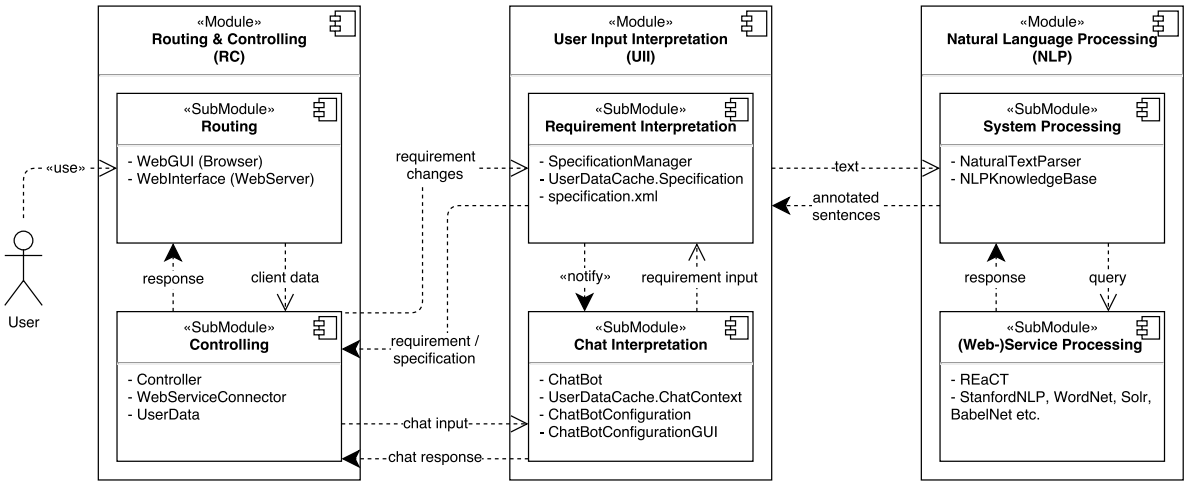

Figure 5. Current conceptual flow of dialog-controlled compensation.

Dialog design aims at the efficient, context-related, comprehensible and target group-oriented dialog connection between humans and computers. This is also the case since a dialog can lead to queries, uncertainties and the need for explanation on the part of the end user. Especially for end users with little prior technical knowledge, it is necessary to lead the iterative clarification processes not only in dialog but also with the help of explanations and examples. This situation-oriented dialog design and user-oriented interaction requires examples and explanations in a knowledge base for retrieval purposes (Section 5.1). For instance, in the compensation of incompleteness, this can mean not only pointing out the lack of information and asking for its compensation, but also providing a similar example that fits into the context and highlighting the concrete information in this example.

5.2.2. CORDULA$_2$: Current Prototype

The revised version of CORDULA aims to overcome the named weaknesses of the existing system (cf. Section 3). This begins with the design of the web interface, which has to be changed so that a dialog between end users and the system is in the center of attention (cf. Figure 6) and also affects fundamental system components such as the knowledge base or the internal communication between the system components. The idea of using a chat bot 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. Figure 6 shows the main chat window of our current prototype, which 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 which provide requirements and additional information. The user input will be forwarded to the server and processed (cf. Figure 5). The server's response will be articulated via the chat bot. Inaccuracies are pointed out and a selection of tailored action proposals is offered. While chatting, the end users can confirm the system's suggestion or reject it. In addition, the requirement can be edited or deleted. As a result, users communicate with the system in two ways. On the one hand, users transfer the requirements via 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 will receive a chat that adapts to the situation.

As shown on the right-hand side of Figure 6, end users are informed what happens with their initial input text and are able to edit the resulting software requirements [37]. Chat bots can thus help within the compensation process and make the results transparent for end users. Of course, the question still remains whether this approach is too complicated for end users who are not technically trained. This will also have to be investigated in further research. In our current prototype of the chat bot, we have also included much information in the front-end, which may need to be reduced in order to not overwhelm the users. This also affects the possibility of modifying domain-oriented information such as POS tags or syntactic structures on the web interface (inline-window below

right). Here, we offer linguistically trained end users the possibility to easily correct errors have occurred in the compensation process of CORDULA before further processing is started in the OTF Computing pipeline. This possibility is difficult to transfer to a chat dialog, since it would be much more complicated to communicate changes of individual tokens or syntactic structures via dialog than to directly apply them with a computer mouse.

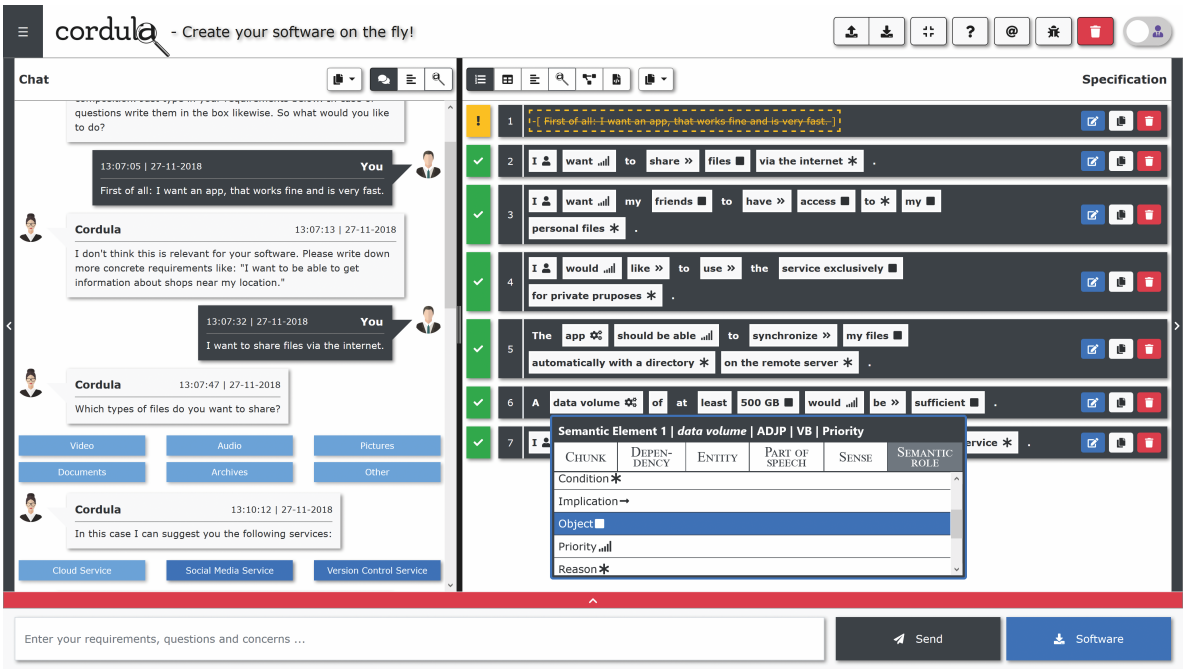

Figure 6. NL software requirement compensation via chat bot [37].

The current prototype can already process software requirements from end users provided via the chat, classify and analyze them for inaccuracies (using the CORDULA back-end [19]) and present them in a structured way. CORDULA$_2$ can also address deficiencies identified in the chat and suggest choices in the event of incompleteness, provided these are defined in the Knowledge Base in a similar context to the identical process word. However, there is still a lot of work to be done in building the Knowledge Base. If it is determined that a certain constellation of detected software requirements refers to an existing software service in the Knowledge Base, CORULDA$_2$ can recommend this service as composition to the end users.

6. Conclusions

The challenges mentioned in this paper can be also found in traditional requirements engineering. Because of the specific nature of OTF Computing, NLP approaches developed so far do not achieve the required execution times. Therefore, there is a lot to be done: attention still needs to be paid to the development of methods for extracting requirements as well as to the detection and compensation of inaccuracies. This raises other issues such as the lack of resources but also the lack of interoperability of individual compensation components, ways of efficiently involving end users without overburdening them, and much more. Currently existing as well as future techniques from the research area of the Semantic Web can be used in the future, in particular the Linked (Open) Data area. The clear referencing of knowledge and the endless expandability of independent knowledge resources is in absolute harmony with the OTF Computing idea—however, improvements can be achieved not only in the integration of external knowledge but also in the modeling of internal OTF processes. Through the discussion (Section 4), we presented the open questions and current challenges for the NLP in the context of OTF Computing and hope to have given some structure to the following research in this area. At the same time, we regard these shortcomings and challenges as a road map for our own

research in the field of NLP pipeline configuration and execution. In particular, we will continue to work on modern chat technology to conduct targeted communication with end users as part of the compensation steps. Through this procedure, missing information can be requested, and the user is supported in answering the questions with examples, etc. Furthermore, the results generated during the compensation process can be explained during the offered chat, which increases the usability for end users. We designed and built a knowledge base that contains over one million example sentences and millions of tokens and relationships among them. This comprehensive resource enables us to provide a vast number of examples to a user in the chat dialog. However, the knowledge base could be further useful if an additional component makes use of the semantics of tokens and builds example services. That is, in a knowledge base, there could be services composed of the requirement descriptions in order to further better assist users in chat bot dialogs. Our future work is dedicated to further utilizing this resource and building new resources that undertake more analyses. As a result, though there are many open challenges, we are well on the way to solving them. Previous research encourages us to walk further on our journey towards making OTF Computing work for everyone.

Author Contributions: Conceptualization, F.S.B.; methodology, F.S.B.; software, F.S.B.; resources, F.S.B. and J.K.; writing—original draft preparation, F.S.B., J.K. and M.G.; writing—review and editing, F.S.B., J.K. and M.G.; visualization, F.S.B. and J.K.; supervision, M.G.; project administration, F.S.B. and M.G.; funding acquisition, M.G.

Funding: This work was partially supported by the German Research Foundation (DFG) within the Collaborative Research Center "On-The-Fly Computing" (SFB 901) under the project number 160364472-SFB901.

Acknowledgments: We thank our student assistant Edwin Friesen for his contribution.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

CORDULA	Compensation of Requirements Descriptions Using Linguistic Analysis
DFG	Deutsche Forschungsgemeinschaft (German Research Foundation)
I/O	Input/Output
NL	Natural Language
NLP	Natural Language Processing
OTF	On-The-Fly
POS	Part-Of-Speech
SFB	Sonderforschungsbereich (Collaborative Research Center)
SRL	Semantic Role Labeling

References

1. Bäumer, F.S.; Geierhos, M. NLP in OTF Computing: Current Approaches and Open Challenges. In *Communications in Computer and Information Science*; Damaševičius, R., Vasiljevienė, G., Eds.; Springer: Berlin/Heidelberg, Germany, 2018; Volume 920, pp. 559–570.
2. Geierhos, M.; Bäumer, F.S. How to Complete Customer Requirements: Using Concept Expansion for Requirement Refinement. In *Proceedings of the 21st NLDB*; Métais, E., Meziane, F., Saraee, M., Sugumaran, V., Vadera, S., Eds.; Springer: Manchester, UK, 2016.
3. Moens, M.F.; Li, J.; Chua, T.S. (Eds.) *Mining User Generated Content*; CRC Press: Leuven, Belgium; Beijing, China; Singapore, 2014.
4. Platenius, M.C.; Josifovska, K.; van Rooijen, L.; Arifulina, S.; Becker, M.; Engels, G.; Schäfer, W. *An Overview of Service Specification Language and Matching in On-The-Fly Computing (v0.3)*; Technical Report Tr-ri-16-349; Software Engineering Group, Heinz Nixdorf Institut, Paderborn University: Paderborn, Germany, 2016.

5. Geierhos, M.; Schulze, S.; Bäumer, F.S. What did you mean? Facing the Challenges of User-generated Software Requirements. In *Proceedings of the 7th ICAART*; Special Session on PUaNLP 2015; Loiseau, S., Filipe, J., Duval, B., van den Herik, J., Eds.; SCITEPRESS—Science and Technology Publications: Lisbon, Portugal, 2015; pp. 277–283.
6. Ferrari, A.; dell' Orletta, F.; Spagnolo, G.O.; Gnesi, S. Measuring and Improving the Completeness of Natural Language Requirements. In *Requirements Engineering: Foundation for Software Quality*; Salinesi, C., van de Weerd, I., Eds.; Springer: Essen, Germany, 2014; Volume 8396, pp. 23–38.
7. Dollmann, M.; Geierhos, M. On- and Off-Topic Classification and Semantic Annotation of User-Generated Software Requirements. In *Proceedings of the Conference on EMNLP*; ACL: Austin, TX, USA, 2016.
8. Bäumer, F.S. Indikatorbasierte Erkennung und Kompensation von Ungenauen und Unvollständig Beschriebenen Softwareanforderungen. Ph.D. Thesis, Paderborn University, Paderborn, Germany, 2017.
9. Pekar, V.; Felderer, M.; Breu, R. Improvement Methods for Software Requirement Specifications: A Mapping Study. In Proceedings of the 9th QUATIC, Guimaraes, Portugal, 23–26 September 2014; pp. 242–245.
10. Umber, A.; Bajwa, I.S. Minimizing Ambiguity in Natural Language Software Requirements Specification. In Proceedings of the 6th ICDIM, Melbourn, VIC, Australia, 26–28 September 2011; pp. 102–107.
11. Kamsties, E. Understanding Ambiguity in Requirements Engineering. In *Engineering and Managing Software Requirements*; Aurum, A., Wohlin, C., Eds.; Springer: Berlin/Heidelberg, Germany, 2005; pp. 245–266.
12. Kamsties, E.; Paech, B. Taming Ambiguity in Natural Language Requirements. In Proceedings of the 13th International Conference on System and Software Engineering and Their Applications (ICSSEA'00), Paris, France, 5–8 December 2000; pp. 1–8.
13. Firesmith, D. Common Requirements Problems, Their Negative Consequences, and the Industry Best Practices to Help Solve Them. *J. Object Technol.* **2007**, *6*, 17–33. [CrossRef]
14. Kamsties, E.; Berry, D.M.; Paech, B. Detecting Ambiguities in Requirements Documents Using Inspections. In Proceedings of the 1st Workshop on Inspection in Software Engineering (WISE'01), Paris, France, 23 July 2001; pp. 68–80.
15. Tichy, W.F.; Landhäußer, M.; Körner, S.J. *nlrpBENCH: A Benchmark for Natural Language Requirements Processing*; Technical Report for RECAA—Requirements Engineering Complete Automation Approach; Karlsruhe Institute of Technology (KIT): Karlsruhe, Germany, 2015.
16. Bäumer, F.S.; Dollmann, M.; Geierhos, M. Studying Software Descriptions in SourceForge and App Stores for a better Understanding of real-life Requirements. In Proceedings of the 2nd ACM SIGSOFT International Workshop on App Market Analytics, Paderborn, Germany, 5 September 2017; Sarro, F., Shihab, E., Nagappan, M., Platenius, M.C., Kaimann, D., Eds.; ACM: New York, NY, USA, 2017; pp. 19–25.
17. Navigli, R.; Ponzetto, S.P. Joining Forces Pays Off: Multilingual Joint Word Sense Disambiguation. In Proceedings of the 2012 Joint Conference on EMNLP and CONLL, Jeju Island, Korea, 12–14 July 2012; pp. 1399–1410.
18. Navigli, R.; Ponzetto, S.P. BabelNet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. In *Artificial Intelligence*; Elsevier: Essex, UK, 2012; Volume 193, pp. 217–250.
19. Bäumer, F.S.; Geierhos, M. 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, Waikoloa Village, HI, USA, 3–6 January 2018; pp. 5746–5755.
20. Dollmann, M. Frag die Anwender: Extraktion und Klassifikation von funktionalen Anforderungen aus User-Generated-Content. Master's Thesis, Paderborn University, Paderborn, Germany, 2016.
21. Vlas, R.; Robinson, W.N. A Rule-Based Natural Language Technique for Requirements Discovery and Classification in Open-Source Software Development Projects. In Proceedings of the 2011 44th Hawaii International Conference on System Sciences (HICSS), Kauai, HI, USA, 4–7 January 2011; pp. 1–10.
22. Körner, S.J. RECAA—Werkzeugunterstützung in der Anforderungserhebung. Ph.D. Thesis, Karlsruher Institut für Technologie, Karlsruhe, Germany, 2014.
23. Huertas, C.; Juárez-Ramírez, R. NLARE, a Natural Language Processing Tool for Automatic Requirements Evaluation. In Proceedings of the CUBE International Information Technology Conference (CUBE'12), Pune, India, 3–5 September 2012; ACM: New York, NY, USA, 2012; pp. 371–378.

24. Fabbrini, F.; Fusani, M.; Gnesi, S.; 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, Greenbelt, MD, USA, 27–29 November 2001; pp. 97–105.
25. Tjong, S.F.; Berry, D.M. The Design of SREE—A Prototype Potential Ambiguity Finder for Requirements Specifications and Lessons Learned. In *Requirements Engineering: Foundation for Software Quality*; Doerr, J., Opdahl, A.L., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 7830, pp. 80–95.
26. Bajwa, I.S.; Lee, M.; Bordbar, B. Resolving Syntactic Ambiguities in Natural Language Specification of Constraints. In *Computational Linguistics and Intelligent Text Processing*; Gelbukh, A., Ed.; Springer: Berlin/Heidelberg, Germany, 2012; Volume 7181, pp. 178–187.
27. Husain, S.; Beg, R. Advances in Ambiguity less NL SRS: A review. In Proceedings of the 2015 IEEE International Conference on Engineering and Technology (ICETECH), Coimbatore, India, 20 March 2015; pp. 221–225.
28. Shah, U.S.; Jinwala, D.C. Resolving Ambiguities in Natural Language Software Requirements: A Comprehensive Survey. *SIGSOFT Softw. Eng. Notes* **2015**, *40*, 1–7. [CrossRef]
29. Bano, M. Addressing the Challenges of Requirements Ambiguity: A Review of Empirical Literature. In Proceedings of the 5th International Workshop on EmpiRE, Ottawa, ON, Canada, 24 August 2015; pp. 21–24.
30. Lami, G. *QuARS: A Tool for Analyzing Requirements*; Technischer Bericht ESC-TR-2005-014; Carnegie Mellon University: Pittsburgh, PA, USA, 2005.
31. Bucchiarone, A.; Gnesi, S.; Fantechi, A.; Trentanni, G. An Experience in Using a Tool for Evaluating a Large Set of Natural Language Requirements. In Proceedings of the 2010 ACM Symposium on Applied Computing (SAC'10), Sierre, Switzerland, 22–26 March 2010; ACM: New York, NY, USA, 2010; pp. 281–286.
32. Huertas, C.; Juárez-Ramírez, R. Towards Assessing The Quality Of Functional Requirements Using English/spanish Controlled Languages and Context Free Grammar. In Proceedings of the 3rd International Conference on DICTAP, Ostrava, Czech Republic, 20–22 November 2013; pp. 234–241.
33. Körner, S.J.; Brumm, T. Natural Language Specification Improvement with Ontologies. *Int. J. Semant. Comput.* **2010**, *3*, 445–470. [CrossRef]
34. Berners-Lee, T.; Hendler, J.; Lassila, O. The semantic web. *Sci. Am.* **2001**, *284*, 28–37. [CrossRef]
35. Piedra, N.; Chicaiza, J.; Lopez-Vargas, J.; Caro, E.T. Guidelines to producing structured interoperable data from Open Access Repositories. In Proceedings of the 2016 IEEE Frontiers in Education Conference (FIE), Erie, PA, USA, 12–15 October 2016; pp. 1–9.
36. Heath, T.; Bizer, C. Linked data: Evolving the web into a global data space. In *Synthesis Lectures on the Semantic Web: Theory and Technology*; Morgan & Claypool Publishers: San Rafael, CA, USA, 2011; Volume 1, pp. 1–136.
37. Friesen, E.; Bäumer, F.S.; Geierhos, M. CORDULA: Software Requirements Extraction Utilizing Chatbot as Communication Interface. In *Joint Proceedings of REFSQ-2018 Workshops, Doctoral Symposium, Live Studies Track, and Poster Track Co-Located with the 23rd International Conference on Requirements Engineering: Foundation for Software Quality (REFSQ 2018)*; Schmid, K., Spoletini, P., Ben Charrada, E., Chisik, Y., Dalpiaz, F., Ferrari, A., Forbrig, P., Franch, X., Kirikova, M., Madhavji, N., et al., Eds.; CEUR Workshop Proceedings (CEUR-WS.org): Essen, Germany, 2018; Volume 2075.
38. Piedra, N.; Tovar, E.; Colomo-Palacios, R.; Lopez-Vargas, J.; Alexandra Chicaiza, J. Consuming and producing linked open data: the case of Opencourseware. *Program* **2014**, *48*, 16–40. [CrossRef]
39. Collobert, R. Deep learning for efficient discriminative parsing. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics (AISTATS), Fort Lauderdale, FL, USA, 11–13 April 2011; Volume 15, pp. 224–232.
40. Collobert, R.; Weston, J.; Bottou, L.; Karlen, M.; Kavukcuoglu, K.; Kuksa, P. Natural language processing (almost) from scratch. *J. Mach. Learn. Res.* **2011**, *12*, 2493–2537.