Integrating a Dialog Component into a Framework for Spoken Language Understanding

Sebastian Weigelt
Karlsruhe Institute of Technology
Karlsruhe, Germany
weigelt@kit.edu

Tobias Hey
Karlsruhe Institute of Technology
Karlsruhe, Germany
hey@kit.edu

Mathias Landhäußer
thingsTHINKING GmbH
Karlsruhe, Germany
mathias@thingsTHINKING.net

ABSTRACT

Spoken language interfaces are the latest trend in human computer interaction. Users enjoy the newly found freedom but developers face an unfamiliar and daunting task. Creating reactive spoken language interfaces requires skills in natural language processing. We show how a developer can integrate a dialog component in a natural language processing system by means of software engineering methods. Our research project PARSE that aims at naturalistic end-user programming in spoken natural language serves as an example. We integrate a dialog component with PARSE without affecting its other components: We modularize the dialog management and introduce dialog acts that bundle a trigger for the dialog and the reaction of the system. We implemented three dialog acts to address the following issues: speech recognition uncertainties, coreference ambiguities, and incomplete conditionals.

We conducted a user study with ten subjects to evaluate our approach. The dialog component achieved resolution rates from 23% to 50% (depending on the dialog act) and introduces a negligible number of errors. We expect the overall performance to increase even further with the implementation of additional dialog acts.

CCS CONCEPTS

• **Human-centered computing → Natural language interfaces**; • **Computing methodologies →** *Lexical semantics*; Natural language processing; *Discourse, dialogue and pragmatics*;

KEYWORDS

Programming in natural language, naturalistic programming, end-user programming, knowledge-based software engineering, natural language processing for software engineering, dialog systems, dialog integration, human-computer interaction

ACM Reference Format:
Sebastian Weigelt, Tobias Hey, and Mathias Landhäußer. 2018. Integrating a Dialog Component into a Framework for Spoken Language Understanding. In *RAISE'18: IEEE/ACM 6th International Workshop on Realizing Artificial Intelligence Synergies in Software Engineering, May 27, 2018, Gothenburg, Sweden.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3194104.3194105

1 INTRODUCTION

Programming should never be a one way street. In traditional software development the compiler (or the IDE) provides the programmer with feedback. When shifting the focus to end-user programming one will notice that such feedback is hardly accessible with a speech interface. Additionally, end-user programming focuses on laypersons. Thus, we can not expect the user to deal with compiler warnings and the like. The most natural way of communication is with a dialog. However, from a developer's point of view building reactive spoken language interfaces requires skills in natural language processing. Even if a natural language processing system is available one has to equip it with interactivity. We show how a developer can integrate a dialog component in an existing natural language processing system by means of software engineering methods. Our research project PARSE that aims at naturalistic end-user programming in spoken natural language serves as an example. *PARSE* provides speech recognition facilities and models the user's intents in a semantic graph; we also use this graph for dialog management.

This paper focuses on how we separate concerns between the framework and the dialog component, i.e., which parts of a dialog system are implemented in the dialog component, which parts are provided by *PARSE*, and how the interaction works. We modularize the dialog management and introduce dialog acts that bundle a problem pattern with a dialog strategy and speech generation templates, i.e., the trigger for the dialog and the desired reaction of the system. dialog acts are organized in a chain of responsibility. This way, we create an order that is extensible at any time with little effort. Even though this case study focuses on *PARSE* we believe that our approach is transferable. To demonstrate our concept, we implemented dialog acts to resolve three issues: speech recognition uncertainties, coreference ambiguities, and incomplete conditionals.

The remainder of the paper is organized as follows. First we discuss related work in section 2. Then we introduce *PARSE* before we present our approach to integrate the dialog component in section 3 and section 4 respectively. In section 5 we take a closer look at the implemented dialog acts before we present our evaluation results in section 6. Section 7 concludes the paper with future work.

2 RELATED WORK

Dialog systems are well studied (c. f. [4]). There are ruled-based and statistical approaches. The former have proven useful in industrial settings such as flight reservation systems or call routing in customer care. The latter are more in the focus of recent research [15].

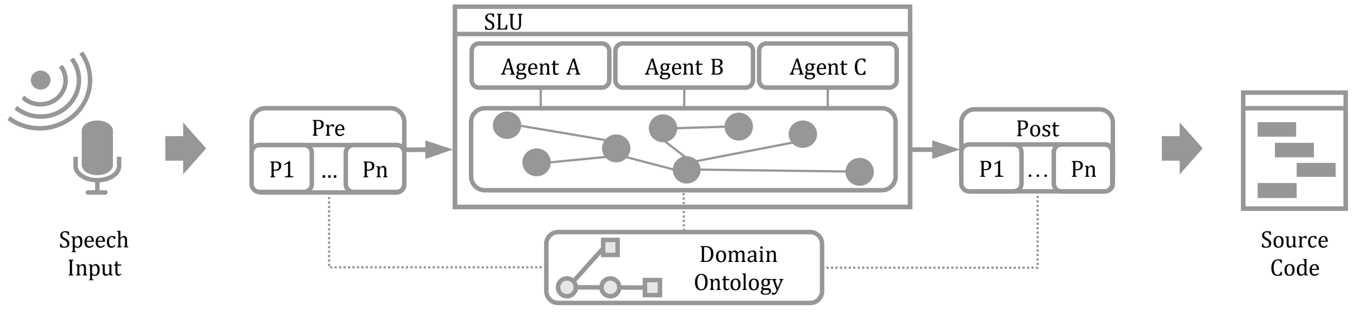

Figure 1: The architecture of PARSE.

Rule-based systems process input in various ways. Some model the decision process as finite state machines (*call-flow based* [7, 9]), some respond according to predefined slots that have to be filled (*frame-based* [12]), others divide dialog management into smaller units that handle a specific task of the dialog (*agenda-based* [1, 11]), or use inference to generate responses (*information-state based* [8]).

Academic research focuses on statistical methods. For instance, Partially Observable Markov Decision Processes (POMDP) are used to formulate dialog management as a planning problem to predict the users' intention [10, 14, 20]. As training data is limited, recent approaches use reinforcement learning to train dialog policies online [3]. However, in POMDP-based dialog systems the dialog state and action space have to be designed with care and the language understanding and generation modules still need corpora for supervised learning. Neural Network approaches push the dialog management away from modeling dialog states explicitly and try to emulate the behavior encountered in the training data [13, 19]. Li et al. [6] even transfer the concept of reinforcement learning to neural networks. Thus, the systems adapts between dialog situations.

3 THE PARSE FRAMEWORK

As mentioned in section 1 we integrate a dialog component into an existing framework. The framework is part of the project *PARSE* [18]. The goal of the project is to enable a layperson to program in plain spoken English. Typical application areas of *PARSE* are robotics, home automation, and the like. To facilitate programming with spoken language the system must understand the user's intents. Thus, *PARSE* is actually a system for Spoken Language Understanding (SLU) [15]. To achieve deep SLU *PARSE* uses an agent-based approach. Every agent is responsible for a specific SLU task. As SLU tasks are interdependent in general, all agents work in parallel and therefore may benefit from results of other agents. The strict separation of concerns enables us to either build a knowledge-based or a probabilistic agent depending on the SLU task at hand and to evaluate it intrinsically. The architecture of *PARSE*, which is illustrated in Figure 1, is separated in three independent parts: a pipeline for pre-processing, an agent-based main execution, and a pipeline for post-processing. A graph serves as shared data structure for the agents. The pre-processing pipeline is meant for common natural language processing (NLP) tasks, e.g., automatic speech recognition (ASR). The user's utterance is processed sequentially here. In the last pre-processing step, the initial graph is built and passed to the SLU module. There, agents work in parallel and transform the graph to publish their results. Hereby, a semantic representation of the input is created incrementally. SLU tasks encompass detection of actions and conditionals [16] or coreference, topic, and context analysis [17]. If the graph cannot be transformed into a proper intent model, the utterance is likely to be incomplete or ambiguous. (In such situations the user should be queried for clarification.) The post-processing pipeline performs the steps required to map the user's intents – modeled in the graph – to functions of the target system. Target systems are modeled in ontologies as proposed in our previous project *NLCI* [5]. We have shown, that domain ontologies can be extracted (semi-)automatically from most APIs with little effort.

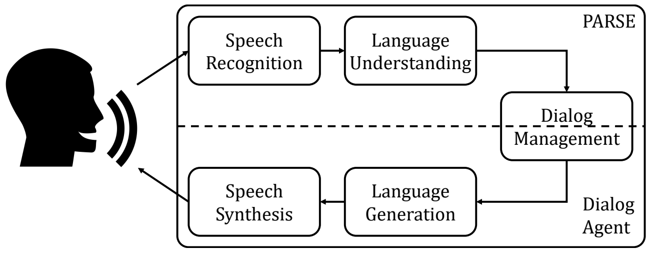

Figure 2: The general architecture of the *PARSE* dialog system, adapted from reference [4]: *PARSE* deals with speech recognition and language understanding while language generation and speech synthesis is part of the dialog agent; dialog state is maintained in *PARSE*'s graph.

4 APPROACH

Figure 2 shows the general architecture of dialog systems. Typically, they encompass modules for speech recognition, language understanding, dialog management, language generation, and speech synthesis. The speech recognition module converts the acoustic signal into a stream of phonemes or words; some speech recognition systems provide alternative transcripts associated with confidence levels. The transcripts are passed to the language understanding module that interprets the utterances. Usually, it transforms the natural language stream into a logical representation. Next, the dialog management module interprets the results of the language understanding module. Often, interpretation draws from the latest input, prior utterances, and – if available – additional information, such as situational context. How this process is carried out is where the various approaches differ the most. But regardless of whether an approach uses recurrent neural networks or flow graphs, the basic idea is as follows: The dialog management has an internal state that is determined by prior input. Incoming utterances modify the internal state and state transitions trigger system reactions. The language generation module produces the required word stream, e.g., a request for missing information. Many approaches use templates for this task. Finally, the speech synthesis module transforms the word stream into an audio signal.

We integrate the dialog component as a new agent for *PARSE*. This way, the dialog agent can use information created by other agents. In the same manner, the other agents may use the results of the dialog for further analyses. If we would integrate the dialog component as pre- or post-processing modules these synergies go missing. The project setting imposes additional challenges:

(1) We expect *PARSE* to resolve most uncertainties by itself; the dialog component must be reactive, i.e., it should observe the other agents' work and start a dialog only if necessary.
(2) *PARSE* can be extended with new agents easily and new agents can introduce new problem classes. New problem classes might require new dialog acts, thus the dialog agent must be extensible too.
(3) End-user programming with spoken language entails long and complex utterances. Consequently, asking a reasonable question is complex. Therefore, we have to carefully design dialog management and language generation.

The following subsections detail how we built the dialog agent.

4.1 Speech Recognition and Language Understanding

As shown in Figure 2, two building blocks of dialog systems (i.e., speech recognition and language understanding) are already present in *PARSE*. The dialog agent leverages them. At present, we are using IBM's ASR *Watson – Speech to Text*[1] for speech recognition.

Understanding spoken language is the core functionality of *PARSE*. As described in section 3, agents work in parallel on *PARSE*'s semantic intent graph. Our dialog agent can use the framework to interpret not only the semantic graph but also the answers to its requests. Every new utterance produces an additional graph which is interpreted by the dialog agent; newly gathered information is fed back to the semantic graph. In doing so, we assure that the semantic graph represents the user's intent for the entire discourse (and not the verbatim answers).

4.2 Dialog Management

The dialog agent monitors *PARSE*'s semantic graph to respond to information needs, uses it to persist its state, and updates it with information gained by interacting with the user. If the semantic graph bears ambiguous or incomplete information a system reaction is triggered. We discuss how we detect such situations in section 5. Hence, the semantic graph serves as a *global* dialog model.

However, this global dialog management is insufficient. Depending on the situation it might be necessary to pose multiple questions to gather needed information from the user: We need to keep track of previous questions to react properly, e.g., to avoid repetition of the same questions or dead end situations. To do that, the dialog agent manages the *local* dialog state itself until the dialog is completed and the resolution is fed back to the semantic graph.

[1]For *Watson Speech to Text* see https://www.ibm.com/watson/services/speech-to-text/. Two additional speech recognizers are being integrated into *PARSE*. It will then process a merged output from all ASRs.

Table 1: Excerpt of identified dialog acts.

Problem class		
Information	**Problem**	**Indicator**
speech recognition	uncertainty	low word confidence (ASR)
		many word alternatives
coreference	ambiguity	similar confidence of potential references
	missing	pronoun without reference
	uncertainty	low confidence of sole reference
conditional	missing	
	missing then-clause	conditional clause misses a then-clause
	missing else-clause	

4.3 Language Generation & Speech Synthesis

Since *PARSE* does not offer feedback mechanism, language generation and speech synthesis are implemented in the dialog agent. Our language generation is frame-based with slots to fill. For any system reaction we designed one or more frames. Following this approach we are able to combine fixed formulation with actual content (words or phrases) representing the context of the question. Several examples are presented in section 5. After all slots of the frames are filled, the question is sent to IBM's *Watson – Text To Speech*[2] for speech synthesis.

4.4 Implementation of the Dialog Agent

The basic structure of our approach is as follows. We use so called dialog acts that react to defined system states, i.e., solve a certain problem class. We organize them in a chain of responsibility [2]. To tackle a new problem class one must implement a dialog act and add it to the chain of responsibility. The chain with its dialog acts structures local dialog management, language generation, and speech synthesis.

Starting with the first dialog act, every act searches for its indicators in the *PARSE* graph. If no pattern matches, the responsibility is passed to the next dialog act. If a pattern matches, the dialog is initiated and the dialog act stays in control until it is finished. Whenever a dialog act finishes, the process starts anew and the chain is traversed from the beginning. Thereby, a successfully concluded dialog act might resolve a problem covered by another dialog act. For example, if a speech recognition error is resolved, a previously incomplete conditional sentence may then be interpreted correctly without questions. If no pattern matches, we assume the utterance to be completely understood and do not ask any questions.

[2] See https://www.ibm.com/watson/services/text-to-speech/.

5 DIALOG ACTS

As discussed in section 4, the dialog agent should be reactive. We use the concept of dialog acts to identify issues a dialog could resolve. In related work (e.g., [4, 15]), dialog acts refer to utterances that require a system reaction. We adapt and extend this concept in the following way. Instead of phrases, we use graph patterns as indicators for a problem class that can be solved by a dialog act. E.g., a low *confidence* attribute of a word node might indicate an ASR error. Similarly, a word node with more than one outgoing *reference* edge indicates a coreference ambiguity.

Additionally, we attach a solution to each dialog act that encompasses a dialog strategy with a suitable questions to ask. All dialog strategies are modeled as finite-state machine. Every state transition triggers a system reaction. The entry action of the final state feeds the information back into the semantic graph. Each state transition has one or more question frames, which are used for language generation.

This definition of dialog acts is the key to extensibility: If a new problem class should be tackled, the dialog agent remains unchanged; instead we simply add a dialog act. Any agent may define a new dialog act.

The problem classes we identified for *PARSE* include missing NER tags, uncertain semantic roles or word senses, and the like. Table 1 lists the problem classes that we chose to implement first: *speech recognition uncertainty*, *coreference ambiguity*, and *incomplete conditionals*. Our selection is based on the following criteria: We need at least one indicator to implement a dialog act. From all acts with an indicator, we chose speech recognition uncertainties because they are hard to resolve without dialog and influence all other analyses. We selected the two additional dialog acts because they are easy to implement, especially regarding dialog management. The following subsections detail the selected dialog acts.

5.1 Speech Recognition Uncertainty

The precision of the ASR is critical, as its outcome influences all following steps: the more errors in the ASR's output, the harder the correct interpretation of the utterance. On the other hand we do not want to bother users with asking too many questions.

Indicator. Table 1 shows that the analysis of *PARSE*'s structure yielded two potential indicators: a low word confidence provided by the ASR and many word alternatives. The latter is inappropriate in general as the number of word alternatives depends largely on the used ASR (which is interchangeable in *PARSE*) and the number of the word's homophones. The number of word alternatives depends largely on the used ASR (which is interchangeable in *PARSE*) and the number of the word's homophones. Thus, we focused on the indicator *low word confidence*. The implementation of this indicator is simple: if the confidence of a word is below a threshold, ask for clarification. A low threshold results in many uncertainties to be cleared up but triggers many questions. Just the opposite is the case with a high threshold. We discuss the impact of the threshold in subsection 6.2.

Dialog Model. Now that we spotted the potentially wrong recognized word, how do we ask for clarification? Simply asking for the word itself could be ambiguous if the word occurs multiple times. The alternatives provided by the ASR are of little value, as asking for homophones, like "Do you mean *write* or *right*" would likely result in no information gain. Therefore, we decided to ask the user to repeat the potentially erroneous part of the utterance. To interpret the response it has to be aligned with the initial utterance. We align the two utterances based on words with a high confidence. Then unsure words in the initial utterance are substituted by their counterparts from second input if their confidence is above the threshold. As an example consider the following ASR output with the recognition confidence in subscript: "… take$_{0.91}$ the$_{0.82}$ right$_{0.72}$ french$_{0.52}$ … ". If the response to the clarification question results in the ASR output "take$_{0.94}$ the$_{0.92}$ white$_{0.85}$ fridge$_{0.81}$" the unsure words *right* and *french* can be replaced with the corrected counterparts. If the response can not be aligned, the question is repeated and the user is asked to use the same wording he used in the initial input. If this fails as well, control is given to the next dialog act in the chain.

Language Generation. As we want the user to repeat a part of the input we use frame-based language generation. The structure of the question is as follows: "Please repeat the following part of your statement: `[pot_err_part]`". Thereby, `pot_err_part` is determined by all words between the beginning of the verb phrase directly before and the end of the noun phrase immediately after the potentially erroneous word(s). For the second question only the static part of the response is extended to remind the user to use the same wording as in the initial utterance.

5.2 Coreference Ambiguity

Coreference resolution can be difficult.Especially in spoken language, clarification is needed to prevent misinterpretations. Consider the example "can you see the cup and plate over there bring *it* to me". The pronoun "it" is referring to one of the two entities mentioned before (cup or plate), but even humans cannot determine which one it refers to.

Indicator. Coreferring expressions and their referents are represented by nodes which are connected by reference edges with a confidence in *PARSE*'s graph. Thus, indicators are based on this information. As pronouns normally refer to another entity the indicator for missing anaphora references are words with POS tag *PRP* (personal pronoun) without outgoing reference edges. We consider all words and phrases with more than one outgoing reference edge (with similar confidence levels) to be ambiguous references. Uncertain coreference information is indicated by a word or phrase that has only one reference edge with a low confidence.

Dialog Model. Asking for coreferences is difficult. In general, users do not know the concept of coreference and simple questions such as, "Does it refer to X?", are not helpful, as *it* and *X* may occur several times in the utterance. Therefore, we repeat the part of the utterance that contains both, the referring expression and the potential referent(s) and ask the user which word the referring expression refers to. The advantage of this approach is that it suits all three problem classes. To interpret the response, it is aligned with the original utterance. If the response can not be aligned we ask the user again and list all potential referent(s). Finally, we ask for every candidate referent consecutively.

Language Generation. To ask the user for the correct referent we extract the referring expression (`ref_ex`) and its potential referents (`list_of_ref`). Then we determine all clauses between these expressions including the clauses of the expressions themselves (`encl_clauses`). We use the frame "In the following, what does the word `[ref_ex]` refer to? `[encl_clauses]`?". For the second try we use a different frame: "I don't understand. You've mentioned `|[list_of_ref]|` entities. Now, please tell me what does the `[ref_ex]` refers to in the following, $\texttt{[list\_of\_ref]}_{0,N-1}$, or $\texttt{[list\_of\_ref]}_{N}$?". If we do not have a list of potential referents, we only ask the first question and use up to two clauses as enclosing clauses.

5.3 Incomplete Conditionals

In previous work, we identified conditionals in the user's utterance and mapped them to if-then-else constructs in the semantic graph [16]. Our analyses are robust regarding speech recognition errors and missing keywords (a *then*-clause is usually not introduced by a keyword). Nevertheless, they are not always correct, e.g. when a keyword was not recognized by the ASR or when the user chooses a sentence structure we do not cover. If we can neither detect the conditional clause itself nor the full extend of one of its dependent clauses, the generated source code is likely to be incorrect.

Indicator. We only identified one indicator for this problem class: the missing dependent then-clause (see Table 1). Given a conditional statement, we can only deduce that a *then*-clause is needed; the *else*-clause is optional. Because of this, we can only ask for missing *then*-clauses. If we identify a conditional clause without a corresponding *then*-clause, we consider this conditional as being incomplete.

Dialog Model. The dialog model for asking the user to add a missing *then* clause is composed of the following steps. First we ask the user to repeat the instruction that frames the *then* clause in the instructions following the conditional clause. If the response can be matched successfully to the original utterance, the information about the detected *then* clause is added to *PARSE*'s graph. If not, the conditional clause is repeated and the user is asked which actions should be triggered when the condition holds. If this question is still unsuccessful, a similar question with different wording is asked. If this fails as well, we cannot resolve this problem.

Language Generation. If we detect a conditional clause (`if_part`) without a dependent *then*-clause, we collect all phrases from the conditional up to the next detected conditional. These phrases are used as `pot_then_part` in the first asked question: "Please just say the then condition of the following part again: `[pot_then_part]`". For the second question we use the frame: "Please just repeat the part, which tells me what I have to do if the condition is true. Your words were: `[pot_then_part]`". The third frame is "Please repeat the instruction in your statement, if the following condition is satisfied: `[if_part]`".

6 USER STUDY

We performed a user study to evaluate our approach. Five of ten subjects were female and five male; they were 22 to 32 years old. All of them are native German speakers but six lived in an English-speaking country for six months or more. All but two described their English proficiency as at least "advanced" (*CEF* level B2).

6.1 Setup

In the user study, we asked the subjects to instruct a domestic robot using *PARSE*. We provided them with three scenarios in each of which the robot performs a task in a kitchen setting. Also, we expected the subjects to answer all clarification questions *PARSE* asked during the interaction. All subjects participated in all three scenarios while we tracked their interaction with the system. In total, we collected 30 protocols.

In the first scenario, the robot is supposed to take the laundry from the washing machine and put it into the dryer. The subjects should instruct the robot verbally and answer any question that arises.

In the second scenario, we provided the subjects with a recorded interaction from a previous user study that was conducted without dialog (c.f. reference [17] for details). The transcript includes anaphoric coreferences that cannot be resolved without additional information. We replayed the recording to *PARSE* and asked the subjects to answer any arising questions. We perform this study because we know the triggered dialog acts in advance (speech recognition uncertainty and coreference ambiguity) and thus can compare the responses of the subjects with each other.

The third scenario, just like the second, uses a recording from another user study. This transcript only triggers dialog acts due to incomplete conditionals (c.f. reference [16]).

The subjects perform the scenarios in the described order: They started with the free dialog and proceeded to the recordings afterwards. Because of that, the subjects were not influenced by the wording of the recordings.

As we want to evaluate the validity of the entire approach, we evaluated the dialog agent with all three dialog acts enabled. Since the dialog acts are ordered in a chain of responsibility, a dialog act completed successfully at the beginning of the chain may render following acts obsolete. Thus, the number of initiated dialog acts per subject differs not only in scenario one but also for scenarios two and three. For the speech recognition uncertainties we measured the success of the dialog and the impact of the ASR confidence threshold. In case of the coreference ambiguities we only measured the success of the dialog and counted the number of questions asked. For the incomplete conditionals we counted the successful dialogs and how often the missing then-clauses could be identified in the responses. The following subsections describe the results of the individual evaluations.

6.2 Speech Recognition Uncertainty

We evaluate speech recognition uncertainties in scenarios one and two. Scenario three is omitted because the transcription does not contain speech recognition errors. To determine a reasonable ASR confidence threshold we assessed how it influences error resolution in scenario one. Furthermore, we count how many uncertainties are resolved (in case of an error) or verified (in case of a low confidence of a correct recognition), how many new errors are introduced, and how many questions it takes to resolve the uncertainty.

Table 2: Impact of different thresholds in the pattern for detecting speech recognition uncertainties.

Threshold	0.70	0.75	0.80	0.85	0.90	0.95
Recall	0.714	**0.730**	0.722	0.655	0.624	0.566
Precision	0.476	0.548	0.619	0.679	0.750	**0.821**
F1	0.571	0.626	0.667	0.667	**0.681**	0.670
Accuracy	0.847	0.860	**0.868**	0.855	0.850	0.827

Table 3: Evaluation results: dialog management for speech recognition uncertainties

	scenario 1	scenario 2
# dialog acts started	44	33
# questions asked	80	58
# dialog acts successful	11	12
# verified utterances	6	11
# resolved errors	4	1
# newly introduced errors	1	0
error rate	0.02	0.00
success rate	0.25	0.36
resolution rate	0.23	0.36

Threshold. To measure the impact of the threshold, we labeled the erroneous words in the transcripts of scenario one. Based on this classification we calculated precision, recall, accuracy and F1-score regarding whether our approach with the given threshold detected them as erroneous. Table 2 shows the results. As expected, precision and recall depend on the threshold: a low threshold results in high recall but low precision; a high threshold results in the opposite. We consider the accuracy to be pleasantly high. The difference to precision and recall can be explained by the fact that true negatives, which are only considered in accuracy, are an order of magnitude more frequent than the other parts. We chose a threshold of 0.8 for the other evaluations since it produced the best accuracy.

Dialog Acts. We measure the success of our dialog approach in scenarios one and two with the following metrics: success rate (how many dialogs end with an update of the semantic graph), resolution rate (how many dialogs actually resolve errors), and error rate (how many dialogs introduced new errors). Table 3 shows the results.

In total 77 words had an ASR confidence below the threshold of 0.8 and led to a dialog. 23 dialog acts were completed successfully, 17 of which led to a verification and five resolved errors. Even though only 29% of the identified errors and uncertainties were resolved or verified, only one additional error was introduced. In total 80 questions for scenario one and 58 questions for scenario two were asked, which means 1.8 questions per dialog act. However, all successfully completed dialog acts were solved with only one question answer pair. In some cases the questions were misinterpreted and the subjects rephrased the phrase or even the whole sentence, which can be seen as one reason that the resolution rate is that low. Another reason is the pronunciation of some subjects, which prevented a successful interpretation of their responses. However, only one new error was introduced.

Table 4: Evaluation results: dialog management for coreference ambiguity.

	you → you	it → green cup	∑
resolution rate	0.40	0.60	0.50
Ø questions	2.00	1.50	1.75

Table 5: Evaluation results: dialog management for incomplete conditionals.

	rate	Ø questions
success	1.00	1.10
resolution	0.30	1.33

6.3 Coreference Ambiguity

To compare the runs of the different subjects, we limit this evaluation to scenario two. The recording contains two coreference ambiguities. The first coreference is an anaphor with the pronoun *you*; it has two possible referents: the previous occurrence of *you* and the proper noun *Philip*. *Philip* is just an erroneous transcription of the phrase *fill it* by the ASR. Thus, the correct referent is the other *you*. The second coreference is the pronoun *it*, which could refer to three entities, namely *green cup*, *table* and *Armar* (the name of the robot in this scenario); the correct referent is *green cup*.

The first ambiguity is a good example for the advantage of the chain of responsibility: If the word error can be resolved by the dialog act for speech recognition uncertainty, the reference can be resolved unambiguously.

The results are shown in Table 4. Again, we consider the resolution rate as in the previous subsection. For the first reference the measured resolution rate is 40% with two questions asked on average. The resolution rate for the second reference is 60% with only 1.5 questions asked on average. If we consider only the subjects who lived in English-speaking region for more than six months, the results are even better: 50% of the first reference and 83% of the second can be resolved in a dialog. This results are surprisingly good considering that this dialog act is complex and it is hard to frame precise questions that can be interpreted easily by users.

6.4 Incomplete Conditionals

We evaluate incomplete conditionals only for scenario three for the same reasons as in the previous evaluation. The recording contains one incomplete conditional. *PARSE* was able to detect a conditional clause, but no dependent clauses.

The results are shown in Table 5. The metrics are defined as previously, now depicting the successfully interpreted user utterances and resolved incomplete conditionals. The success rate of 100% means that all subjects were able to answer the questions of the dialog agent. However, only 30% of the subjects described the dependent clause as we expected. This outcome illustrates how hard it is to pose a question that instructs the user properly in how he or she should answer. In the remaining 70% the subjects often repeated the entire conditional which results in the conditional clause to be treated as part of the *then* clause as well. This time there was only a minor difference between novices and advanced

English speakers. We believe that this hints at a more general issue. In the future we will avoid such misinterpretations by ignoring the conditional clause in the answer.

6.5 Discussion

Our evaluation does not cover all aspects yet. With the evaluation setup, we do our best not to influence the subjects in the way they interact with the system. The interaction of our subjects is influenced by their English proficiency but six out of ten spent six months or more in an English-speaking country. Given the results of our case study, we expect the performance of our approach to increase with the users' language proficiency. Also, we ran the framework in *production mode* and do not evaluate the dialog acts individually; we believe that this approach reflects real-world performance in the best way possible. Finally, we did not evaluate the implementation complexity of new dialog acts as neither lines of code nor spent time is informative. Even though no changes to the framework were necessary, we cannot tell whether this holds for future dialog acts.

7 CONCLUSION & FUTURE WORK

We presented how we integrated a dialog component into an existing framework for SLU. The framework we consider is *PARSE*, which aims at end-user programming in natural language. As language understanding is not perfect, a means for feedback and a way of solving natural language issues is necessary.

PARSE can be extended easily with new components that provide new capabilities but can introduce new issues as well. We implemented dialog acts that represent problem classes with their respective indicators and resolution strategies. The dialog acts are organized in a chain of responsibility and thus can be extended with little effort. When a new component is introduced, its developer can define new dialog acts to counteract new issues or gather more input. To demonstrate our approach, we implemented three dialog acts covering the problem classes *Speech Recognition Uncertainty*, *Coreference Ambiguity* and *Incomplete Conditionals*.

We integrated the dialog component as a new agent for *PARSE*. This way, it can leverage *PARSE*'s NLU components. Global dialog state is stored in the semantic graph that is maintained by *PARSE* and issue resolutions are fed back to it immediately. Thereby, the other agents profit directly from additional information.

We evaluated our approach in a user study where we let ten subjects interact with *PARSE* in three scenarios each. The dialog component achieved problem resolution rates between 23% and 50% and introduced few errors. For subjects that had lived in an English-speaking country the results are even better; for these subjects the dialog component achieves a resolution rate of up to 67%.

We believe that our approach can be applied to any NLU system that provides the means for global dialog management. Therefore, user utterances must be represented as a semantic representation (or interlingua). If no such representation is provided, the dialog management has to be done completely in the dialog component. In that case, we believe that a dialog-centered system is advisable.

In future work we will extend the number of dialog acts to cover more language understanding issues. Up to now, we have identified 14 different dialog acts spanning from uncertain Part-of-Speech tags to missing or unclear semantic role labels. Also, we want to make the definition of dialog acts easier so that additional problem classes can be covered with ease.

REFERENCES

[1] Dan Bohus and Alexander I. Rudnicky. 2003. RavenClaw : Dialog Management Using Hierarchical Task Decomposition and an Expectation Agenda. *EUROSPEECH-2003* (2003).

[2] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software.* Pearson Education.

[3] M. Gašić, D. Kim, P. Tsiakoulis, C. Breslin, M. Henderson, M. Szummer, B. Thomson, and S. Young. 2014. Incremental On-Line Adaptation of POMDP-Based Dialogue Managers to Extended Domains. In *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH.*

[4] Kristiina Jokinen and Michael McTear. 2009. Spoken Dialogue Systems. *Synthesis Lectures on Human Language Technologies* 2, 1 (Jan. 2009), 1–151. https://doi.org/10.2200/S00204ED1V01Y200910HLT005

[5] Mathias Landhäußer, Sebastian Weigelt, and Walter F. Tichy. 2016. NLCI: A Natural Language Command Interpreter. *Automated Software Engineering* (2016). https://doi.org/10.1007/s10515-016-0202-1

[6] Jiwei Li, Will Monroe, Alan Ritter, Michel Galley, Jianfeng Gao, and Dan Jurafsky. 2016. Deep Reinforcement Learning for Dialogue Generation. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing.* ACL, Austin, Texas, 1192–1202.

[7] Michael F. McTear. 1998. Modelling Spoken Dialogues with State Transition Diagrams: Experiences with the CSLU Toolkit. In *5th International Conference on Spoken Language Processing (ICSLP 98).* Sydney, Australia.

[8] Fabrizio Morbini, David DeVault, Kenji Sagae, Jillian Gerten, Angela Nazarian, and David Traum. 2014. FLoReS: A Forward Looking, Reward Seeking, Dialogue Manager. In *Natural Interaction with Robots, Knowbots and Smartphones.* Springer.

[9] R. Pieraccini, S. Caskey, K. Dayanidhi, B. Carpenter, and M. Phillips. 2001. ETUDE, a Recursive Dialog Manager with Embedded User Interface Patterns. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 2001.* 244–247. https://doi.org/10.1109/ASRU.2001.1034633

[10] Nicholas Roy, Joelle Pineau, and Sebastian Thrun. 2000. Spoken Dialogue Management Using Probabilistic Reasoning. In *Proceedings of the 38th Annual Meeting on Association for Computational Linguistics (ACL '00).* Association for Computational Linguistics, Stroudsburg, PA, USA, 93–100. https://doi.org/10.3115/1075218.1075231

[11] Alexander Rudnicky and Wei Xu. 1999. An Agenda-Based Dialog Management Architecture for Spoken Language Systems. In *IEEE Workshop on Automatic Speech Recognition and Understanding, 1999. ASRU '99.* 1–337.

[12] Stephanie Seneff and Joseph Polifroni. 2000. Dialogue Management in the Mercury Flight Reservation System. In *Proceedings of the 2000 ANLP/NAACL Workshop on Conversational Systems - Volume 3 (ANLP/NAACL-ConvSyst '00).* Association for Computational Linguistics, Stroudsburg, PA, USA, 11–16. https://doi.org/10.3115/1117562.1117565

[13] Iulian V. Serban, Alessandro Sordoni, Yoshua Bengio, Aaron Courville, and Joelle Pineau. 2016. Building End-to-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI'16).* AAAI Press, Phoenix, Arizona, 3776–3783.

[14] B. Thomson, J. Schatzmann, and S. Young. 2008. Bayesian Update of Dialogue State for Robust Dialogue Systems. In *2008 IEEE International Conference on Acoustics, Speech and Signal Processing.* https://doi.org/10.1109/ICASSP.2008.4518765

[15] Gokhan Tur and Renato De Mori. 2011. *Spoken Language Understanding: Systems for Extracting Semantic Information from Speech.* Wiley.

[16] Sebastian Weigelt, Tobias Hey, and Vanessa Steurer. 2018. Detection of Conditionals in Spoken Utterances. In *2018 IEEE 12th International Conference on Semantic Computing (ICSC).*

[17] Sebastian Weigelt, Tobias Hey, and Walter F. Tichy. 2017. Context Model Acquisition from Spoken Utterances. In *The 29th International Conference on Software Engineering & Knowledge Engineering,.* Pittsburgh, PA, 201–206. https://doi.org/10.18293/SEKE2017-083

[18] Sebastian Weigelt and Walter F. Tichy. 2015. Poster: ProNat: An Agent-Based System Design for Programming in Spoken Natural Language. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering (ICSE)*, Vol. 2. 819–820. https://doi.org/10.1109/ICSE.2015.264

[19] Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gašić, Lina M. Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 2016. A Network-Based End-to-End Trainable Task-Oriented Dialogue System. In *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics.* ACL, 438–449.

[20] Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. 2010. The Hidden Information State Model: A Practical Framework for POMDP-Based Spoken Dialogue Management. *Computer Speech & Language* 24, 2 (April 2010). https://doi.org/10.1016/j.csl.2009.04.001