ADVISER: A Dialog System Framework for Education & Research

Daniel Ortega, Dirk Väth, Gianna Weber, Lindsey Vanderlyn,
Maximilian Schmidt, Moritz Völkel, Zorica Karacevic and Ngoc Thang Vu*
Institute for Natural Language Processing (IMS), University of Stuttgart
thangvu@ims.uni-stuttgart.de

Abstract

In this paper, we present ADVISER[1] - an open source dialog system framework for education and research purposes. This system supports multi-domain task-oriented conversations in two languages. It additionally provides a flexible architecture in which modules can be arbitrarily combined or exchanged - allowing for easy switching between rules-based and neural network based implementations. Furthermore, ADVISER offers a transparent, user-friendly framework designed for interdisciplinary collaboration: from a flexible back end, allowing easy integration of new features, to an intuitive graphical user interface supporting non-technical users.

1 Introduction

Dialog systems can be open-ended, e.g. small talk systems (Weizenbaum, 1966), or designed for a specific task, such as booking flights or finding restaurants (Bobrow et al., 1977; Wen et al., 2017). In this paper, we focus on task-oriented dialog systems, although our framework allows easy integration of non-task dialog systems (Vinyals and Le, 2015) and their combination (Yu et al., 2017). Task-oriented dialog systems are generally comprised of sequential modules addressing the varying subtasks required to facilitate a natural language dialog (Williams et al., 2016). A standard architecture implements this with a natural language understanding (NLU) unit which is responsible for parsing the user input (De Mori et al., 2008), a belief state tracker (BST) module (Mrkšić et al., 2017) that holds the state of the dialog, a policy module (Williams and Young, 2007) which determines the system's next action, and a natural language generation (NLG) module which transforms the system act into natural language. Recently, there has been increasing interest in multi-domain, task-oriented dialog systems because of their ability to help users achieve more complex goals, which may not be cleanly divided into single-domain objectives (Mrkšić et al., 2015; Ultes et al., 2017). This increased freedom for users, however, requires increasingly sophisticated system architectures to effectively handle cross-domain actions or transitions between active domains.

During the last years, several toolkits (Bohus and Rudnicky, 2009; Skantze and Moubayed, 2012; Baumann and Schlangen, 2012; Lison and Kennington, 2016; Ultes et al., 2017; Miller et al., 2017) have been introduced to accelerate the development and testing process of goal-oriented dialog systems, for both single-domain and multi-domain systems. Almost all of them have been developed for fast prototyping, where new domains can be developed by swapping in new implementations for each module following the toolkit's architecture. However, generating natural and effective dialogs requires linguistic knowledge to be integrated throughout the system design, and most of these toolkits are primarily designed for technical users, which can limit the ease of collaboration with linguists and may thus affect the quality of the system's output.

To address these shortcomings, we propose a multilingual multi-domain dialog system with two parallel goals: 1) to provide a highly flexible research framework not only for technique oriented developers but also for non-technical oriented developers such as linguists and 2) to provide an interdisciplinary educational tool.

[1]Website: https://www.ims.uni-stuttgart.de/forschung/ressourcen/werkzeuge/adviser.html

* All authors contributed equally.

Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics: System Demonstrations, pages 93–98
Florence, Italy, July 28 - August 2, 2019.

2 Related Work

During the last decade, several toolkits have been developed to facilitate the rapid implementation of goal-oriented dialog systems. RavenClaw (Bohus and Rudnicky, 2009) aimed to separate domain-specific aspects from domain-independent conversational skills, letting developers focus solely on describing the dialog task control logic. However, the most recent techniques like a statistical BST or a reinforcement learning (RL)-based policy are not compatible.

Our approach is inline with InproTK (Baumann and Schlangen, 2012), where high-level modules (such as NLU and dialog manager) communicate by networks created via configuration files. In ADVISER, there is no distinction between high or low-level modules, but they are similarly designed separately and our `DialogSystem` class defines the pipeline interaction between them, allowing them to easily be replaced with newer versions.

OpenDial (Lison and Kennington, 2016) relies on probabilistic rules and easily integrates external modules. We were inspired by this work to make the rules-based dialog systems fully domain-independent in ADVISER by storing the NLU rules and NLG templates in external files.

PyDial (Ultes et al., 2017), a multi-domain dialog toolkit, follows the classic approach for modular dialog systems. Although PyDial offers rules-based and statistical implementations for the modules, the overall structure is rather difficult to manipulate, if the pipeline needs to be modified.

3 Framework Design

Our goal with this system is to provide both a highly modular research platform and an interdisciplinary educational tool.

Use Cases To accomplish this, we address the needs of the following three user groups: 1) technical users such as machine learning researchers 2) non-technical researchers such as linguists and 3) multidisciplinary students.

Design Criteria The main *objectives* of the framework are threefold: 1) to maximise the ease for technical developers when exploring new architectures or extending system functionality with new techniques, e.g. machine learning. 2) To minimise the workload on code bases for non-technical users (e.g. linguists) allowing them to focus on their main goals, e.g. exploring the dialog flow for new domains, or languages or investigating human language variations when interacting with a dialog system. 3) To provide an engaging way for multidisciplinary students to learn how dialog systems work.

From this, our framework is designed to optimise the following criteria:

Modularity: For each module in a classic dialog system pipeline (NLU, BST, dialog policy and NLG), we provide a handcrafted baseline module, additionally we provide a machine learning based implementation for the BST and policy (see section 4.2). These can be used to quickly assemble a working dialog system or as implementation guidelines for custom modules. Additionally, because all modules inherit from the same abstract class, technical users can also easily write their own implementations or combinations of modules.

Flexibility: In contrast to a more static dialog system pipeline, we propose a graph structure where the user is in control of the modules and their order. This level of control allows users to realise anything from pipelines to end-to-end systems. Even branching scenarios are possible as demonstrated by our meta policy which combines multiple parallel subgraphs into a single dialog.

Transparency: Inputs to and outputs from each module are captured by automatically generated XML interface descriptions, providing a transparent view of data flow through the dialog system.

User-friendly at different levels: technical users have the full flexibility to explore and extend the back end; non-technical users can use our defined modules for building systems; students from different disciplines could easily learn the concepts and explore human machine interaction.

4 Modules

4.1 Graphical User Interface

Graphical user interfaces (GUIs) allow users to access a system in an easy, clear and appealing fashion. Thus, in addition to a console, ADVISER provides two separate graphical interfaces: a GUI to chat with the dialog system and a gamelike interface for study purposes.

Chat interface Our GUI is implemented as a module, which is called by the dialog system once at the beginning and once at the end of each turn. In the first turn, the GUI is initialised and loaded.

At the beginning of each turn, it blocks the processing pipeline until the user has entered a message. The message is displayed inside the GUI and then passed to succeeding modules, e.g. the NLU module. At the end of the turn, the module takes the output of the NLG and displays it inside the GUI.

Gamelike interface for study purposes Handcrafted NLU modules are often based on regular expressions (regexes), which aim to find patterns inside a user utterance in order to identify possible user acts. The module's developers try to cover as many user act realisations (UARs) as possible. However, due to the versatile nature of human language, many regexes are needed to yield a high coverage. In ADVISER, we provide an interface which supports collaboration between computer scientists and linguists to yield a higher quality of the NLU module. To motivate both sides, we frame this challenge as a game - the CrossTick game - in which computer scientists try to achieve high regex coverage and linguists try to write uncovered UARs. First, the user has to select the domain for which UARs are written and the NLU module that should be evaluated. After a UAR is created, it is analysed by the specified module and the user is informed via a tick (✓) or a cross (×) whether the user acts were detected correctly. The user can save and load files in JSON format.

4.2 Components of Dialog Systems

Input Up to now, only text is supported but our tool could be easily extendable to other modalities such as speech and vision. Currently text can either be entered through the console or our GUI.

NLU We implemented a domain-independent rules-based NLU that loads regexes from a JSON file. The regexes are split into three categories - general acts (e.g. *Hello*, *RequestAlternatives* and *Affirm*), domain-specific inform acts and domain-specific request acts. We supported both, English and German rules. The NLU module receives the user input as string and checks it across all regexes, creating a list of possible user acts. If no act is found, then it is assumed that the NLU was not capable of understanding and the user act is interpreted as a *BadAct*. We additionally resolve some ambiguities using the belief state, i.e. the dialog history. If a non-contextualised *Affirm* or *Deny* act is found, the system attempts to use the dialog history to contextualise it.

BST The belief state tracker maintains a representation of the current dialog state. The rules-based BST receives a list of user acts from the NLU that are decoded and stored with probabilities in the belief state. The BST also detects the presence of discourse acts, e.g. *Hello*, *Repeat*, *Inform* and *Request*. Moreover, it stores information from the system history including the last requested slot and last entity offered.

Our machine learning based belief state tracker is trained to predict the belief state directly from text without the need for an NLU. To track the constraints and requests issued by the user, we feed system actions and user input turn-wise into a recurrent network and concatenate the resulting hidden states of both inputs before predicting the final belief state (Jagfeld and Vu, 2017).

Policy The rules-based policy aims to provide users with a single entity matching the constraints they have specified. After each turn, the policy verifies that the user has not ended the dialog. It then reads the current belief state and generates a suitable query for the database. If there are multiple results, the next system act will request more information from the user to disambiguate. Otherwise, the system is able to make an offer – directly informing the user about a specific entity – or to give more details about a current offer.

Our machine learning policy is trained using deep RL. Similar to the Deep Q-learning algorithm (Mnih et al., 2013), an action-value function is approximated by a neural network which outputs a value for each possible system action given the vectorised representation of a turn's belief state as input. The neural network is constructed following the duelling architecture (Wang et al., 2016), consisting of two separate calculation streams. Each stream has its own layers, where one stream calculates the value function and the other an advantage function so that their combination in a special final layer yields the action-value function again. Additionally, an updated copy of this network is used to evaluate the agent's actions while the original up-to-date network is accessed to choose the agent's greedy actions (Van Hasselt et al., 2016). For the agent's efficient off-policy batch-training, we make use of prioritised experience replay (Schaul et al., 2015) by assigning experienced dialog turns a sampling probability proportional to errors in the action-value estimates.

NLG In the natural language generation module, the semantic representation of the system act is transformed into natural language. In the handcrafted NLG module, each possible system act is mapped to exactly one utterance. To reduce the potentially large number of mappings, templates are used which allow multiple mappings from system acts to their respective utterance at once. By specifying placeholders for a system acts slots and/or values, the utterance can be formulated independent of the actual realisations (e.g. `inform(name={X}, ects={Y})` → `The course {X} is worth {Y} ECTS.`). During the dialog, the system iterates through the templates and chooses the first one for which the system act fits the template's signature. For each domain we present here, we created both German and English templates.

User Simulator To support automatic evaluation and enable RL, we implemented a user simulator to provide user actions at the intention level. For this purpose, we integrated the Agenda-based (Schatzmann et al., 2007) user simulator into our framework. The task of the system is to fulfil the user's goal within the course of the dialog. For this purpose, we populated our agendas with actions requesting information and informing about the constraints based on the specified goal. For the design of a user simulator, we additionally considered that its objective was not only to fulfil the user's goal but also to support the RL policy in the learning process. Therefore, it was not sufficient to answer the system's request and fulfil the user's goal, but also to force the system to answer with suitable actions. In the context of RL, we achieved this by delaying a dialog turn if the system's answer was not an expected action or completely nonsense, because it reduces the reward the policy receives for the ongoing dialog. In addition, several parameters influence the user simulator's behaviour. Those are manually crafted and chosen with a suitable probability wherever variation to the user makes sense.

Meta Policy In a multi-domain dialog system, intelligently switching between or combining individual domains is necessary to provide the user a unified experience. However, the best way to accomplish this remains unclear. In our system, we propose an architecture where all domains are allowed to run in parallel and the resulting output is processed by a Meta Policy. The meta policy is responsible for tracking which domains are active and, if necessary, combining their output. In the case where a user utterance cannot be directly handled in the context of a single domain, the meta policy is also responsible for rewriting it into one or more single-domain utterances. If this happens, rather than outputting something for that turn or asking for user input, the system steps through an additional turn, using the rewritten utterance as the new user input. In this way the meta policy is able to intelligently coordinate switching or combining domains, preserving as much information as possible to make as informed of a decision as possible. This architecture can be seen in figure 1.

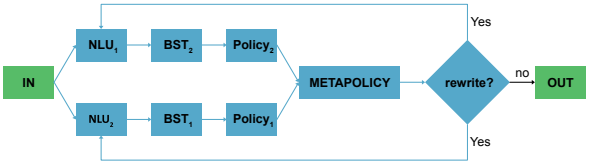

Figure 1: Multi-domain system architecture, showing how information is passed between each module and the final system output is coordinated by the meta policy.

5 Proof-of-Concept Showcases

In order to feedback on the quality, functionality, and usefulness of the ADVISER system, we conducted two experiments: we first investigated user experiences with a student support dialog system and second explored the effectiveness of using a game within a multidisciplinary practical course.

5.1 Student Support Dialog System

Multilingual and multi-domain As a real-world use case, we implemented a dialog system, using our ADVISER framework, to help students navigate through the course and module selection at the Institute for Natural Language Processing (IMS) at the University of Stuttgart. This task consists of three domains for asking information about lecturers, for locating courses, and for collecting information about modules. Students can freely switch between or combine the domains in order to find the information they need. Additionally because of the students' backgrounds, our system supports NLU and NLG in both, English and German.

Evaluation and Results We evaluated our dialog policies automatically on two domains: the new domain - IMS modules and the benchmark domain - Cambridge restaurants (Ultes et al., 2017). Table 1 shows the performance of the supported policies tested against the user simulator. Each result was obtained by averaging across ten different random seeds with 500 test dialogs each.

Agent	CamRestaurants		IMS modules	
	Suc.	Turns	Suc.	Turns
RL	97.5%	5.03	91.6%	5.25
HDC	100%	4.75	99.8%	7.0

Table 1: Percentage of successful dialogs and average dialog length for Cambridge restaurants and IMS modules domains evaluated with a RL policy and a handcrafted policy (HDC).

To gain a better understanding of how ADVISER works under real-world conditions, we asked 13 volunteers to conduct a series of five conversations with the system. Participants were required to chat with ADVISER via text input in English to find modules offered by the IMS. For each dialog, participants were given a list of goals, detailing constraints to inform the system about and to request from the system. These included information such as responsible lecturer, the term within which the module was offered, and whether the module was related to a particular discipline, e.g. linguistics and statistics.

After each dialog, participants were asked to rate their chat with ADVISER, considering the quality of the respective dialog in terms of naturalness and coherence as well as how successful ADVISER was in processing the information provided by the user and how comfortable it was to use. Overall, participants rated the quality of dialog with the RL policy a 4.3 similar to the results obtained from dialogs using the rules-based policy (4.48 points on average) on a scale from 1 (very bad) to 7 (very good), confirming the functionality of the system. Considering the system's success in processing the user information correctly, on a scale from 1 (very bad) to 5 (very good), participants again found ADVISER to be slightly above average. This result applies to both dialogs which were generated using the rules-based policy (3.52 on average) and those using the RL policy (3.52 point average). 61.5 % of the participants reported that they would use ADVISER for their own purposes. Moreover, asking participants to rate the comfort of using the system on a scale from 1 (very bad) to 5 (very good), an average comfort level was 3.69, indicating that most users felt comfortable with the system.

5.2 Multidisciplinary Practical Course

To test ADVISER as a tool for study purposes, we evaluated the CrossTick game, where linguistics and computer science students work to develop and gain a better understanding of the NLU in a dialog system.

CrossTick game Given a user act, linguistics students/researchers work to find as many UARs as possible which are not recognised by the system. On the other side, computer science students/researchers minimise the system errors by developing either new rules or machine learning models to handle more natural language variations of the user inputs.

Evaluation In order to test the efficiency and the user's opinion about the game, the same 13 participants from the previous evaluation were given 10 tasks. Per task, users were asked to take the linguist's perspective, and given intent, slots, and values to generate natural language for. In this case *intent* corresponds to the type of sentence (e.g. *Inform*/*Request*), *slot* to the type of information a user is giving/requesting and *value* describes the actual information given. If participants succeeded in creating natural language variations that were not covered by the system's NLU, they saw a cross mark ($\times$) next to their input, and points were added to their score. Otherwise, they obtained a check mark ($\checkmark$), worth 0 points. Although users could reach an infinite number of points per task, they were encouraged to be more productive and creative by telling them to beat the high score another user previously scored.

During the survey, 84.6% of the participants stated that the game was effective for educational purposes. Further, on a scale from 1 (completely useless) to 5 (very useful) the CrossTick Game received on average 3.69 points, suggesting that most users learned the NLU module's functionality. Overall, participants enjoyed using the game. Some participants especially liked the challenge to beat the high score, while others enjoyed that they were rewarded with points for uncovered sentences.

6 Conclusions

In this paper, we presented ADVISER - an open source dialog system which supports multilingual, multi-domain human-machine task-oriented conversations. It supports modules which can easily be interchanged between rules-based and machine learning implementations –including deep learning and RL. Our preliminary human study shows that with our toolkit one can easily build a useful dialog system. Furthermore, the CrossTick game offers an appealing interface for education purposes for different study disciplines.

Acknowledgments

We would like to thank all the voluntary students at the University of Stuttgart for their participation in the evaluation. This work was funded by the Carl Zeiss Foundation.

References

Timo Baumann and David Schlangen. 2012. The inprotk 2012 release. In *NAACL-HLT Workshop on Future Directions and Needs in the Spoken Dialog Community: Tools and Data*.

Daniel G. Bobrow, Ronald M. Kaplan, Martin Kay, Donald A. Norman, Henry Thompson, and Terry Winograd. 1977. GUS: a Frame-Driven Dialog System. *Artificial Intelligence*, 8.

Dan Bohus and Alexander I. Rudnicky. 2009. The ravenclaw dialog management framework: Architecture and systems. *Comput. Speech Lang.*

Renato De Mori, Frédéric Bechet, Dilek Hakkani-Tur, Michael McTear, Giuseppe Riccardi, and Gokhan Tur. 2008. Spoken language understanding. *IEEE Signal Processing Magazine*.

Glorianna Jagfeld and Ngoc Thang Vu. 2017. Encoding word confusion networks with recurrent neural networks for dialog state tracking. In *Proceedings of the Workshop on Speech-Centric Natural Language Processing*.

Pierre Lison and Casey Kennington. 2016. Opendial: A toolkit for developing spoken dialogue systems with probabilistic rules. In *Proceedings of ACL*.

Alexander Miller, Will Feng, Dhruv Batra, Antoine Bordes, Adam Fisch, Jiasen Lu, Devi Parikh, and Jason Weston. 2017. Parlai: A dialog research software platform. In *Proceedings of EMNLP*.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. 2013. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gasic, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2015. Multi-domain dialog state tracking using recurrent neural networks. In *Proceedings of ACL*.

Nikola Mrkšić, Diarmuid O Séaghdha, Tsung-Hsien Wen, Blaise Thomson, and Steve Young. 2017. Neural belief tracker: Data-driven dialogue state tracking. In *Proceedings of ACL*.

Jost Schatzmann, Blaise Thomson, Karl Weilhammer, Hui Ye, and Steve Young. 2007. Agenda-based user simulation for bootstrapping a pomdp dialogue system. In *Proceedings of NAACL*.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. 2015. Prioritized experience replay. In *Proceedings of ICLR*.

Gabriel Skantze and Samer Al Moubayed. 2012. Iristk: a statechart-based toolkit for multi-party face-to-face interaction. In *ICMI*.

Stefan Ultes, Lina M. Rojas Barahona, Pei-Hao Su, David Vandyke, Dongho Kim, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gasic, and Steve Young. 2017. PyDial: A Multi-domain Statistical Dialogue System Toolkit. In *Proceedings of ACL*.

Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of AAAI*.

Oriol Vinyals and Quoc Le. 2015. A neural conversational model. In *Proceedings of ICML*.

Ziyu Wang, Tom Schaul, Matteo Hessel, Hado Van Hasselt, Marc Lanctot, and Nando De Freitas. 2016. Dueling network architectures for deep reinforcement learning. In *Proceedings of ICML*.

Joseph Weizenbaum. 1966. ELIZA: A Computer Program for the Study of Natural Language Communication Between Man and Machine. *Communications of the ACM*, 9(1).

Tsung-Hsien Wen, David Vandyke, Nikola Mrkšić, Milica Gašić, Lina M Rojas-Barahona, Pei-Hao Su, Stefan Ultes, and Steve Young. 2017. A network-based end-to-end trainable task-oriented dialogue system. In *Proceedings of EACL*.

Jason Williams, Antoine Raux, and Matthew Henderson. 2016. The dialog state tracking challenge series: A review. *Dialogue & Discourse*.

Jason Williams and Steve Young. 2007. Partially observable markov decision processes for spoken dialog systems. *Computer Speech & Language*.

Zhou Yu, Alexander Rudnicky, and Alan Black. 2017. Learning conversational systems that interleave task and non-task content. In *Proceedings of IJCAI*.