PyDial: A Multi-domain Statistical Dialogue System Toolkit

Stefan Ultes, Lina Rojas-Barahona, Pei-Hao Su, David Vandyke*, Dongho Kim†, Iñigo Casanueva, Paweł Budzianowski, Nikola Mrkšić, Tsung-Hsien Wen, Milica Gašić and Steve Young
Cambridge University Engineering Department, Trumpington Street, Cambridge, UK
{su259,lmr46,phs26,ic340,pfb30,nm480,thw28,mg436,sjy11}@cam.ac.uk

Abstract

Statistical Spoken Dialogue Systems have been around for many years. However, access to these systems has always been difficult as there is still no publicly available end-to-end system implementation. To alleviate this, we present PyDial, an open-source end-to-end statistical spoken dialogue system toolkit which provides implementations of statistical approaches for all dialogue system modules. Moreover, it has been extended to provide multi-domain conversational functionality. It offers easy configuration, easy extensibility, and domain-independent implementations of the respective dialogue system modules. The toolkit is available for download under the Apache 2.0 license.

1 Introduction

Designing speech interfaces to machines has been a focus of research for many years. These Spoken Dialogue Systems (SDSs) are typically based on a modular architecture consisting of input processing modules speech recognition and semantic decoding, dialogue management modules belief tracking and policy, and output processing modules language generation and speech synthesis (see Fig. 1).

Statistical SDS are speech interfaces where all SDS modules are based on statistical models learned from data (in contrast to hand-crafted rules). Examples of statistical approaches to various components of a dialogue system can be found in (Levin and Pieraccini, 1997; Jurafsky and Martin, 2008; De Mori et al., 2008; Thomson and Young, 2010; Lemon and Pietquin, 2012; Young et al., 2013; Wen et al., 2015; Su et al., 2016; Wen et al., 2017; Mrkšić et al., 2017).

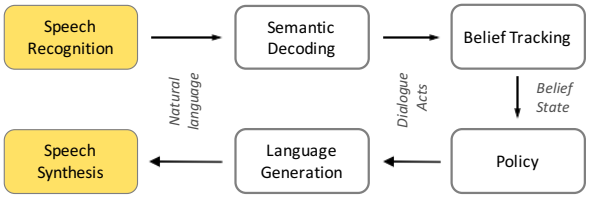

Figure 1: Architecture of a modular Spoken Dialoug System.

Despite the rich body of research on statistical SDS, there is still no common platform or open toolkit available. Other toolkit implementations usually focus on single modules (e.g. (Williams et al., 2010; Ultes and Minker, 2014) or are not full-blown statistical systems (e.g. (Lison and Kennington, 2016; Bohus and Rudnicky, 2009)). The availability of a toolkit targetted specifically at statistical dialogue systems would enable people new to the field would be able to get involved more easily, results to be compared more easily, and researchers to focus on their specific research questions instead of re-implementing algorithms (e.g., evaluating understanding or generation components in an interaction).

Hence, to stimulate research and make it easy for people to get involved in statistical spoken dialogue systems, we present PyDial, a multi-domain statistical spoken dialogue system toolkit. PyDial is implemented in Python and is actively used by the Cambridge Dialogue Systems Group.

PyDial supports multi-domain applications in which a conversation may range over a number of different topics. This introduces a variety of new research issues including generalised belief tracking (Mrkšić et al., 2015; Lee and Stent, 2016) rapid policy adaptation and parallel learning (Gašić et al., 2015a,b) and natural language generation (Wen et al., 2016).

* now with Apple Inc., Cambridge, UK
† now with PROWLER.io Limited, Cambridge, UK

Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics-System Demonstrations, pages 73–78
Vancouver, Canada, July 30 - August 4, 2017.
https://doi.org/10.18653/v1/P17-4013

The remainder of the paper is organized as follows: in Section 2, the general architecture of PyDial is presented along with the extension of the SDS architecture to multiple domains and PyDial's key application principles. Section 3 contains details of the implemented dialogue system modules. The available domains are listed in Section 4 out of which two are used for the example interactions in Section 5. Finally, Section 6 summarizes the key contributions of this toolkit.

2 PyDial Architecture

This section presents the architecture of PyDial and the way it interfaces to its environment. Subsequently, the extension of single-domain functionality to enable conversations over multiple domains is described. Finally, we discuss the three key principles underlying PyDial design.

2.1 General System Architecture

The general architecture of PyDial is shown in Figure 2. The main component is called Agent which resides at the core of the system. It encapsulates all dialogue system modules to enable text-based interaction, i.e. typed (as opposed to spoken) input and output. The dialogue system modules rely on the domain specification defined by an Ontology. For interacting with its environment, PyDial offers three interfaces: the Dialogue Server, which allows spoken interaction, the Texthub, which allows typed interaction, and the User Simulation system. The performance of the interaction is monitored by the Evaluation component.

The **Agent** is responsible for the dialogue interaction. Hence, the internal architecture is similar to the architecture presented in Figure 1. The pipeline contains the dialogue system modules semantic parser, which transforms textual input to a semantic representation, the belief tracker, which is responsible for maintaining the internal dialogue state representation called the belief state, the policy, which maps the belief state to a suitable system dialogue act, and the semantic output, which transforms the system dialogue act to a textual representation. For multi-domain functionality, a topic tracker is needed whose functionality will be explained in Section 2.2. The Agent also maintains the dialogue sessions, i.e., ensures that each input is routed to the correct dialogue. Thus, multiple dialogues may be supported by instantiating multiple agents.

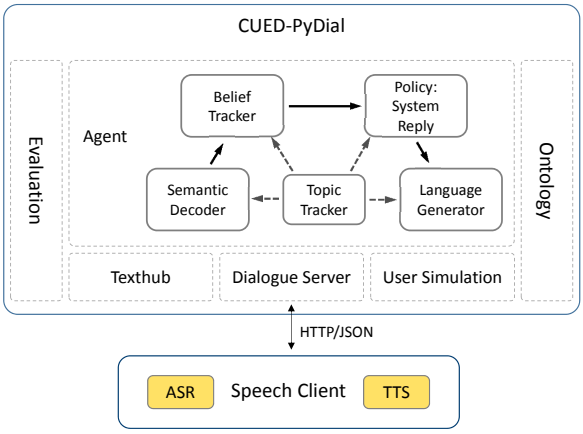

Figure 2: The general architecture of PyDial: the Agent resides at the core and the interfaces Texthub, Dialogue Server and User Simulation provide the link to the environment.

The **User Simulation** component provides simulation of dialogues on the semantic level, i.e., not using any semantic parser or language generation. This is a widely used technique for training and evaluating reinforcement learning-based algorithms since it avoids the need for costly data collection exercises and user trials. It does of course provide only an approximation to real user behaviour, so results obtained through simulation should be viewed with caution!

To enable the Agent to communicate with its environment, PyDial offers two modes: speech and text. As the native interface of the PyDial is text-based, the **Texthub** simply connects the Agent to a terminal. To enable speech-based dialogue, the **Dialogue Server** allows connecting to an external speech client. This client is responsible for mapping the input speech signal to text using Automatic Speech Recognition (ASR) and for mapping the output text to speech (TTS) using speech synthesis. The speech client connects to the Dialogue Server via HTTP exchanging JSON messages. Note that the speech client is not part of PyDial. Cloud-based services for ASR and TTS are widely available from providers like Google[1], Microsoft[2], or IBM[3]. PyDial is currently connected to DialPort (Zhao et al., 2016) allowing speech-based interaction.

Alongside the agent and the interface compo-

[1] https://cloud.google.com/speech
[2] https://www.microsoft.com/cognitive-services/en-us/speech-api
[3] http://www.ibm.com/watson/developercloud/speech-to-text.html

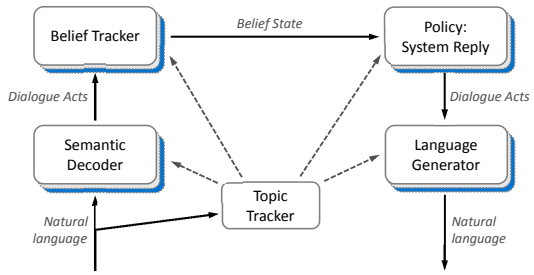

Figure 3: The multi-domain dialogue system architecture: for each module there is an instance for each domain. During runtime, a topic tracker identifies the domain of the current input which is then delegated to the respective domain-pipeline.

nents resides the **Ontology** which encapsulates the dialogue domain specification as well as the access to the back-end data base, e.g., set of restaurants and their properties. Modelled as a global object, it is used by most dialogue system modules and the user simulator for obtaining the relevant information about user actions, slots, slot values, and system actions.

The **Evaluation** component is used to compute evaluation measures for the dialogues, e.g., Task Success. For dialogue modules based on Reinforcement Learning, the Evaluation component is also responsible for providing the reward.

2.2 Multi-domain Dialogue System Architecture

One of the main aims of PyDial is to enable conversations ranging over multiple domains. To achieve this, modifications to the single-domain dialogue system pipeline are necessary. Note that the current multi-domain architecture as shown in Figure 3 assumes that each user input belongs to exactly one domain and that only the user is allowed to switch domains.

To identify the domain the user input or the current sub-dialogue belongs to, a module called the **Topic Tracker** is provided. Based on the identified domain, domain-specific instances of each dialogue module are loaded. For example, if the domain *CamRestaurants* is found, the dialogue pipeline consists of the *CamRestaurants*-instances of the semantic decoder, the belief tracker, the policy, and the language generator.

To handle the various domain instances, every module type has a Manager which stores all of the domain-specific instances in a dictionary-like structure. These instances are only created once for each domain (and each agent). Subsequent inquiries to the same domain are then handled by the same instances.

2.3 Key Principles

To allow PyDial to be applied to new problems easily, the PyDial architecture is designed to support three key principles:

Domain Independence Wherever possible, the implementation of the dialogue modules is kept separate from the domain specification. Thus, the main functionality is domain independent, i.e., by simply using a different domain specification, simulated dialogues using belief tracker and policy are possible. To achieve this, the Ontology handles all domain-related functionality and is accessible system-wide.

While this is completely true for the belief tracker, the policy, and the user simulator, the semantic decoder and the language generator inevitably have some domain-dependency and each needs domain-specific models to be loaded.

Easy Configurability To use PyDial, all relevant functionality can be controlled via a configuration file. This specifies the domains of the conversation, the variant of each domain module, which is used in the pipeline, and its parameters. For example, to use a hand-crafted policy in the domain *CamRestaurants*, a configuration section `[policy_CamRestaurants]` with the entry `policytype = hdc` is used. The configuration file is then loaded by Pydial and the resulting configuration object is globally accessible.

Extensibility One additional benefit of introducing the manager concept described in Sec. 2.2 is to allow for easy extensibility. As shown with the example in Figure 4, each manager contains a set of D domain instances. The class of each domain instance inherits from the interface class and must implement all of its interface methods.

To add a new module, the respective class simply needs to adhere to the required interface definition. To use it in the running system, the configuration parameter may simply point to the new class, e.g., `policytype = policy.HDCPolicy.HDCPolicy`. The following modules and components support this functionality: Topic Tracker, Semantic Decoder, Belief Tracker, Policy, Language Generator, and Evaluation. Since the configuration file is a simple text file, new entries can be added easily using

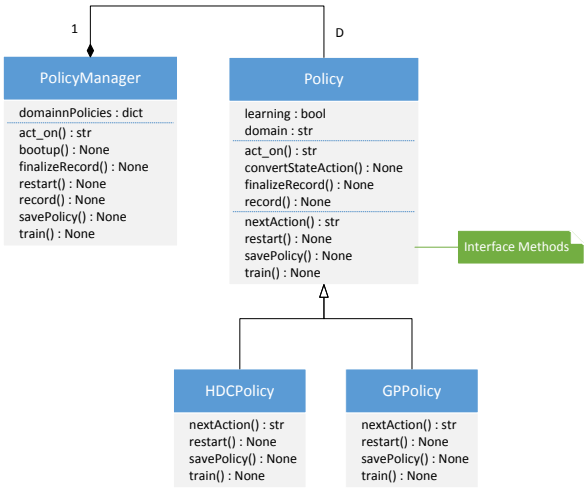

Figure 4: The UML diagram of the policy module. The interface class defines the interface methods for each policy implementation as well as general behaviour relevant for all types of policies. The sub-classes only have to implement the required interface methods. All other modules of the agent have a similar manager-interface architecture.

a convenient text editor and any special configuration options can easily be added.

To add a new domain, a simulated interaction is already possible simply by defining the ontology along with the database. For text-based interaction, an additional understanding and generation component is necessary.

3 Implementation

The PyDial toolkit is a research system under continuous development. It is available for free download from http://pydial.org under the Apache 2.0 license[4]. The following implementations of the various system modules are available in the initial release, however, more will appear in due course.

Semantic Decoder To semantically decode the input sentence (or n-best-list of sentences), PyDial offers a rule-based implementation using regular expressions and a statistical model based on Support Vector Machines, the Semantic Tuple Classifier (Mairesse et al., 2009). For the latter, a model for the *CamRestaurants* domain is provided.

Belief Tracker For tracking the belief state, the rule-based focus tracker is available (Henderson et al., 2014). The implementation is domain-independent. All domain-specific information is drawn from the ontology.

[4] www.apache.org/licenses/LICENSE-2.0

Policy The decision making module responsible for the policy has two implementations: a hand-crafted policy (which should work with any domain) and a Gaussian process (GP) reinforcement-learning policy (Gašić and Young, 2014). For multi-domain dialogue, the policy may be handled like all other modules by a policy manager. Given the domain of each user input, the respective domain policy will be selected.

Additionally, a Bayesian committee machine (BCM) as proposed in Gašić et al. (2015b) is available as an alternative handler: when processing the belief state of one domain, the policies of other domains are consulted to select the final system action. For this to work, the belief state is mapped to an abstract representation which then allows all policies to access it. Within PyDial, trained policies may be moved between the committee-based handler and the standard policy manager handler, i.e., policies trained outside of the committee (in a single- or multi-domain setting) may be used within the committee and vice versa.

Language Generator For mapping the semantic system action to text, PyDial offers two module implementations. For all domains, rule definitions for a template-based language generation are provided. In addition, the LSTM-based language generator as proposed by Wen et al. (2015) is included along with a pre-trained model for the *CamRestaurants* domain.

Topic Tracker PyDial provides an implementation of a keyword-based topic tracker. If the topic tracker has identified a domain for some user input, it will continue with that domain until a new domain is identified. Hence not every user input must contain relevant keywords. If the topic tracker is not able to initially identify the domain, it creates its own meta-dialogue with the user until the initial domain has been identified or a maximum of number of retries has been reached.

Evaluation To evaluate the dialogues, there are currently two success-based modules implemented. The objective task success evaluator compares the constraints and requests the system identifies with the true values. The latter may either be derived from the user simulator or, in real dialogues, by specifying a predefined task. For real dialogues, a subjective task success evaluator may also be applied which queries the user about the outcome of the dialogue.

User Simulation The implementation of the simulated user uses the agenda-based user simulator (Schatzmann et al., 2006). The simulator contains the user model and an error model thus creating a n-best-list of user acts to simulate the noisy speech channel. By using a set of generally applicable parameters, the simulator may be applied for all domains. The domain-specific information is taken from the ontology.

4 Domains

The main focus of PyDial is task-oriented dialogue where the user has to find a matching entity based on a number of constraints. Once the entity is found, the user can request additional information. For this scenario, PyDial is pre-loaded with a total of ten domains of differing complexity:

Domain	Constraints*	Requests†	Entities
CamRestaurants	3	9	110
CamHotels	5	11	33
CamAttractions	3	9	79
CamShops	2	6	21
CamTransport	2	6	5
SFRestaurants	6	11	271
SFHotels	6	10	182
Laptops6	6	10	123
Laptops11	11	21	123
TV	6	14	94

* *# data base search constraints,* † *# information the user can request*

As earlier stated, all policy implementations as well as the belief tracker and the user simulator implementations are independent of the domain. Thus, simulated interactions are possible over all domains. Furthermore, the semantic decoder and the language generator depend to some degree on domain-specific implementations. PyDial contains rule-based semantic decoders for all domains and a statistical decoder for *CamRestaurants*. Also, PyDial comprises template-based language generation for most domains and a statistical LSTM-based generator for *CamRestaurants*. Thus, a fully statistical conversation is possible in the *CamRestaurants* domain.

5 Examples

To illustrate the functionality of PyDial, a GP policy in the *CamRestaurants* and one in the *SFRestaurants* domain is trained in simulation using the objective success evaluator. The final reward is +20 if the dialogue was successful minus the total number of dialogue turns. After 1,000 training dialogues, the policies achieve a task success rate of 95.4% for *CamRestaurants* and 92.0% for *SFRestaurants* when evaluated with 1,000 additional dialogues.

An example multi-domain text-based interaction in the domains *SFRestaurants* and *SFHotels* is shown in the following:

S: Hello, welcome to the San Francisco restaurant and hotel dialogue system. How may I help you?
U: I am looking for a hotel where dogs are not allowed
S: Would you like a hotel that has internet?
U: i don't care
S: Where would you like it to be near to?
U: a cheap hotel in the area of downtown
S: Would you like a hotel that accepts credit cards?
U: i don't care
S: grant hotel is a nice place, it is in the area of downtown , where no dogs are allowed , it is in the inexpensive price range , it has internet and it accepts credit cards .
U: I am also looking for a cheap vegetarian restaurant
S: What part of town do you have in mind?
U: I don't care but it should be for lunch
S: Would you like something in the cheap, moderate, or expensive price range?
U: cheap
S: What kind of food would you like?
U: vegetarian
S: What part of town do you have in mind?
U: I don't care
S: Lucky creation restaurant is a nice place, it is good for lunch , it is in the cheap price range , it serves vegetarian food and it is in the area of chinatown .
U: bye
S: Goodbye.

6 Conclusion

We have presented PyDial, an open-source multi-domain statistical dialogue system toolkit implemented in Python. To facilitate research on statistical dialogue systems, it aims to provide a simple and accessible programming environment for implementing and evaluating new algorithms. Following the key principles of domain-independence, configurability and extensibility, PyDial is built around a modular architecture enabling end-to-end interaction using text or speech input. The toolkit offers example implementations of state-of-the-art statistical dialogue modules and the capability for conversing over multiple domains within a single dialogue.

Source code and documentation

The PyDial source code, step-by-step tutorials and the latest updates can be found on http://pydial.org. This research was funded by the EPSRC grant EP/M018946/1 *Open Domain Statistical Spoken Dialogue Systems*.

References

Dan Bohus and Alexander I Rudnicky. 2009. The ravenclaw dialog management framework: Architecture and systems. *Computer Speech & Language* 23(3):332–361.

R De Mori, F Bechet, D Hakkani-Tur, M McTear, G Riccardi, and G Tur. 2008. Spoken language understanding. *IEEE Signal Processing Magazine* 25(3):50–58.

Milica Gašić, Dongho Kim, Pirros Tsiakoulis, and Steve Young. 2015a. Distributed dialogue policies for multi-domain statistical dialogue management. In *Proceedings of ICASSP*. IEEE, pages 5371–5375.

Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, Tsung-Hsien Wen, and Steve J. Young. 2015b. Policy committee for adaptation in multi-domain spoken dialogue systems. In *Proceedings of ASRU*. pages 806–812. https://doi.org/10.1109/asru.2015.7404871.

Milica Gašić and Steve J. Young. 2014. Gaussian processes for POMDP-based dialogue manager optimization. *IEEE/ACM Transactions on Audio, Speech, and Language Processing* 22(1):28–40.

Matthew Henderson, Blaise Thomson, and Jason Williams. 2014. The second dialog state tracking challenge. In *Proceedings of SIGDial*.

Daniel Jurafsky and James H. Martin. 2008. *Speech and Language Processing*. Prentice Hall, 2 edition.

Sungjin Lee and Amanda Stent. 2016. Task lineages: Dialog state tracking for flexible interaction. In *Proceedings of SIGDial*. ACL, Los Angeles, pages 11–21. http://www.aclweb.org/anthology/W16-3602.

Oliver Lemon and Olivier Pietquin. 2012. *Data-Driven Methods for Adaptive Spoken Dialogue Systems*. Springer New York. https://doi.org/10.1007/978-1-4614-4803-7.

Esther Levin and Roberto Pieraccini. 1997. A stochastic model of computer-human interaction for learning dialogue strategies. In *Eurospeech*. volume 97, pages 1883–1886.

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

François Mairesse, Milica Gasic, Filip Jurcícek, Simon Keizer, Blaise Thomson, Kai Yu, and Steve Young. 2009. Spoken language understanding from unaligned data using discriminative classification models. In *Proceedings of ICASSP*. IEEE, pages 4749–4752.

Nikola Mrkšić, Diarmuid Ó Séaghdha, Blaise Thomson, Milica Gašić, 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*. http://www.aclweb.org/anthology/P15-2130.

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

Jost Schatzmann, Karl Weilhammer, Matt N. Stuttle, and Steve J. Young. 2006. A survey of statistical user simulation techniques for reinforcement-learning of dialogue management strategies. *The Knowledge Engineering Review* 21(2):97–126.

Pei-Hao Su, Milica Gašić, Nikola Mrkšić, Lina Rojas-Barahona, Stefan Ultes, David Vandyke, Tsung-Hsien Wen, and Steve Young. 2016. On-line active reward learning for policy optimisation in spoken dialogue systems. In *Proceedings of ACL*.

Blaise Thomson and Steve J. Young. 2010. Bayesian update of dialogue state: A POMDP framework for spoken dialogue systems. *Computer Speech & Language* 24(4):562–588.

Stefan Ultes and Wolfgang Minker. 2014. Managing adaptive spoken dialogue for intelligent environments. *Journal of Ambient Intelligence and Smart Environments* 6(5):523–539. https://doi.org/10.3233/ais-140275.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Lina M. Rojas-Barahona, Pei-Hao Su, David Vandyke, and Steve Young. 2016. Multi-domain neural network language generation for spoken dialogue systems. In *Proceedings of NAACL-HLT*. http://www.aclweb.org/anthology/N16-1015.

Tsung-Hsien Wen, Milica Gašić, Nikola Mrkšić, Pei-Hao Su, David Vandyke, and Steve Young. 2015. Semantically conditioned lstm-based natural language generation for spoken dialogue systems. In *Proceedings of EMNLP*. ACL, Lisbon, Portugal, pages 1711–1721. https://aclweb.org/anthology/D/D15/D15-1199.

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 D Williams, Iker Arizmendi, and Alistair Conkie. 2010. Demonstration of at&t let's go: A production-grade statistical spoken dialog system. In *Proceedings of SLT*. IEEE, pages 157–158.

Steve J. Young, Milica Gašić, Blaise Thomson, and Jason D. Williams. 2013. POMDP-based statistical spoken dialog systems: A review. *Proceedings of the IEEE* 101(5):1160–1179.

Tiancheng Zhao, Kyusong Lee, and Maxine Eskenazi. 2016. Dialport: Connecting the spoken dialog research community to real user data. In *IEEE Workshop on Spoken Language Technology*.