RETICO: An incremental framework for spoken dialogue systems

Thilo Michael
Quality and Usability Lab
Technische Universität Berlin
thilo.michael@tu-berlin.de

Abstract

In this paper, we present the newest version of retico - a python-based incremental dialogue framework to create state-of-the-art spoken dialogue systems and simulations. Retico provides a range of incremental modules that are based on services like Google ASR, Google TTS, and Rasa NLU. Incremental networks can be created either in code or with a graphical user interface. In this demo, we present three example systems that are implemented in retico: a spoken translation tool that translates speech in real-time, a conversation simulation that models turn-taking, and a spoken dialogue restaurant information service.

1 Introduction

Classical architectures of spoken dialogue systems rely on a pipeline approach, where data is passed through and transformed by a set of modules. These modules perform a specific task on the data, for example, convert speech signals into text (ASR modules) or extracting domain-specific information from text (NLU modules). While this architecture separates the concern between the modules and modularizes the development of spoken dialogue systems, the resulting agents are slow to process data and cannot react quickly to changes in the input.

Incremental processing, an architecture where modules work on small increments of data and forward hypotheses based on those increments to the next module, increases the processing speed and reactiveness of dialogue systems while still retaining the modularized approach of the pipeline architecture (Schlangen and Skantze, 2011). However, due to the overhead of creating and revoking hypotheses and processing on incomplete data, the complexity of each module in an incremental dialogue system increases. For researchers, it can be a challenge to implement and evaluate incremental modules, as they do not have the time and knowledge to implement a complete incremental dialogue system, just to evaluate the part they are researching.

The incremental processing toolkit (InproTK) is an open-source toolkit written in Java that provides an interface for incremental modules and defines an architecture for incremental units, hypothesis handling, and connections between incremental modules (Baumann and Schlangen, 2012). However, the toolkit does not provide an integrated framework that allows for the design and evaluation of networks.

In this paper, we present the current version of retico, an incremental framework for spoken dialogue that was first published in (Michael and Möller, 2019). Retico is a framework written in python and published as an open-source project[1]. We demonstrate three types of speech and dialogue systems that are implemented in this framework. First, we showcase an incremental translation service that utilizes Google Translate to recognize, translate, and synthesize speech. Also, we showcase a simulation of a conversation with turn-taking, where two agents interact with each other. Finally, we showcase a spoken dialogue system in the restaurant information domain. All demo systems are visualized in a graphical user interface, and the networks can be adjusted live (e.g., speech synthesis modules can be switched).

2 Related Work

The incremental model has been formalized by Schlangen and Skantze in (Schlangen and Skantze, 2009, 2011). The resulting framework InproTK (Baumann and Schlangen, 2012) has been used for incremental speech recognition and syn-

[1] Available at www.github.com/uhlo/retico

, pages 154 - 157

thesis and dialogue systems, among others. Based on this, InproTK$_S$ extends the toolkit for the use of situated dialogue(Kennington et al., 2014).

Recent work in modules of spoken dialogue systems like speech recognition (Selfridge et al., 2011) and end-of-turn prediction (Skantze, 2017) focused on incremental processing, and a state-of-the-art natural language understanding module has been incrementalized (Rafla and Kennington, 2019).

Incremental processing cannot only be used in spoken dialogue system, but it also can be useful for research regarding conversation simulation (Michael and Möller, 2020).

3 Architecture

The architecture of retico is written in python based on the conceptual model of incremental processing described in (Schlangen and Skantze, 2009). Core of this framework are the abstract definitions of an *incremental module* (IM) and an *incremental unit* (IU). Both definitions provide interfaces and processing routines to handle concurrent processing of modules and the flow of IUs between the modules. Each IM has a *left buffer*, where IUs of other modules are placed to be processed and a *right buffer* where new hypotheses are placed and sent to IMs further down the incremental pipeline. Usually, an IM defines one or more types of IU that it is able to process and one type of IU it produces and produces or revokes hypotheses based on every incoming unit.

Besides these modules, retico provides interfaces for information exchange apart from IUs by changing meta-information of IMs and by calling "*trigger*" modules, that produce IUs on-demand and insert them to the buffers of modules.

3.1 Incremental Units

Incremental units are mainly defined by their payload, which differs widely depending on the type of data the IU is carrying. For example, an `AudioIU` stores chunks of audio data that is captured by a microphone, while a `TextIU` stores text recognized by an ASR module or generated by an NLG module.

IUs also manage references to IUs they are based on, as well as IUs that precede it. This information is automatically collected and added to the IU when it is created as part of the processing routine of an Incremental Module.

Additionally, IUs retain information on their hypothesis-status, that is, if they are *committed* (no further changes to the hypothesis will be made) or if they are *revoked* (the hypothesis is no longer valid and may be replaced with a newer hypothesis). Also, meta-data in the form of key-value-pairs can be attached to an IU. In contrast to the payload of an IU, the meta-data is not standardized for a type of an IU and is not guaranteed to be present. However, it is a useful tool for debugging or storing information used for visualization.

3.2 Incremental Modules

Incremental modules represent the core functionality of retico. Their connectivity is defined by one or more input IU types and one output IU type. However, there are special *producer* modules that do not accept any input IUs because they obtain information from other sources (e.g., the `MicrophoneModule`) and *consumer* modules that do not output any IUs (e.g., the `SpeakerModule`). The primary processing method of an incremental module is invoked every time there is a new IU in the left buffer, and it may return a new IU for the right buffer. Like IUs, incremental modules are also able to hold meta-data, which is used for debugging and visualization purposes.

Retico already includes modules from various fields of a spoken dialogue system. Most notably, there exists modules that handle Audio input and output, online and offline speech recognition (CMUSphinx, Google ASR), natural language understanding (rasa NLU), dialogue management (agenda-based, rasa RNN-based, n-gram-based), speech synthesis (Mary TTS, Google TTS) as well as translation services (Google Translate). Additional modules and integrations from other frameworks are in planning.

3.3 Logging and Persistence

The IUs that are defined in retico generalize via the python inheritance structure, so that standard data types like audio, text, and dialogue acts are supported. This allows retico to persist IUs of these types with so-called "recorder" modules.

Retico modules are serializable so that networks can be stored into a file to be loaded and initialized again later.

3.4 Graphical User Interface

While modules can be created, connected, and run purely in python code, it also provides a GUI that

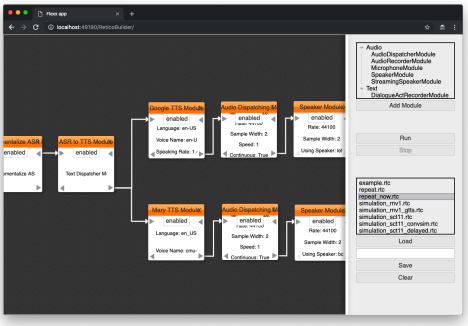

Figure 1: Screenshot of the graphical user interface to create, save and load incremental networks.

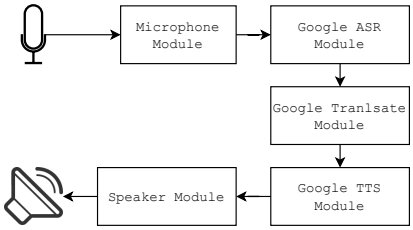

Figure 2: The schema of the translation service.

visualized networks and the flow of IUs. Figure 1 shows the user interface that runs in a browser. Available modules are shown in a tree list and can added onto the canvas. Connections between modules can be made by clicking on the sockets, and the interface highlights only modules that can process the specific IU type. When a network is run in the GUI, the modules show basic information about the IU they are currently processing.

Networks created with the GUI can be saved to a file and be loaded again with the GUI or with python code. The position and size information of the modules are stored in the module's metadata, which allows retico to retain the layout of the network when loading it from file.

4 Demonstrations

In this section, we present three different projects that are created entirely in retico. Due to the modular approach of retico, these systems are not fixed regarding the modules they use for a given task. For example, rectico is able to use two different speech synthesis modules that can be interchanged.

4.1 Spoken Translation Service

The translation service utilizes speech recognition, a text translation service, and speech synthesis to translate sentences spoken into the system. As can be seen in Figure 2, the main components used in this setup are the Google ASR, Google TTS, and the Google Translate modules. While the ASR

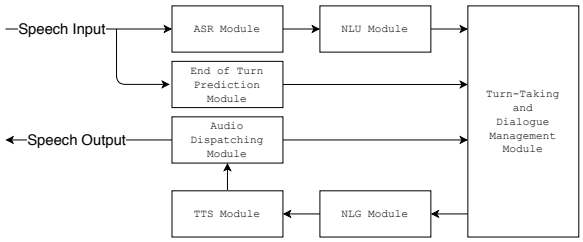

Figure 3: The schema of one agent in the turn-taking conversation simulation network. The complete simulation includes two agents whose speech-input and -output are connected, as well as a recording modules that stores the conversation onto disk for further evaluations.

module works on word and sub-word level, the translation module collects multiple words so that a potential translation stabilizes. The translated sentences are synthesized with Google TTS and transmitted to the speakers.

The languages that can be translated by this service are only limited by the capabilities of Google ASR, TTS, and translation services. However, we tested the system with German-English, English-French, and German-French translations.

Because there is no echo suppression implemented in this version of the service, the loudspeaker, and the microphone have to be acoustically separated (e.g., via a headset).

4.2 Conversation Simulation

The conversation simulation consists of two spoken dialogue systems that are connected and can communicate through an audio channel. Because of the incremental implementation, the agents can predict the end-of-turn of their interlocutors and perform rudimentary turn-taking. As can be seen in Figure 3, an `Audio Dispatching Module` is used to control when an agent speaks and when it is silent, and it also provides feedback of the status of the current utterance back to the dialogue manager. The simulated conversation itself models a short conversation test as standardized by (ITU-T Recommendation P.805, 2007). Concretely, the scenario describes a telephone conversation between a worker at a pizzeria and a customer. The customer inquires about available dishes and their toppings, selects an item from the menu, and the pizzeria worker requests information like telephone number and address.

The modules in this network (ASR, TTS, NLG, NLU, end-of-turn) are based on recorded data from real conversations performed in laboratory conditions that were transcribed and annotated

with dialogue acts and turn markers. The incremental modules in the simulation make use of meta-information transmitted through retico's side-channel to perform their tasks. The utterances produced by the agents are sliced from the empirical conversations. However, other synthesis methods can be used.

4.3 Restaurant Information System

The restaurant information system is a spoken dialogue system that finds restaurants based on user-given criteria like location of the restaurant, as well as the type and price of food. Once every slot is filled, the dialogue system queries a database and recommends restaurants that match the criteria. Depending on the complexity of the query, the request to the database can be slow. The incremental processing, together with a caching-mechanism implemented into the database connector, allows for faster response times of the dialogue system.

The speech recognition and synthesis are realized with Google ASR, and TTS modules, and rasa NLU is used for the natural language understanding. The dialogue manager used in this system is rule-based and uses slot-filling to query restaurants.

5 Conclusion

In this paper, we presented the newest version of retico, a framework for incremental dialogue processing. We described the incremental architecture and highlighted the logging and persistence features as well as the graphical user interface. We also showcased three application ideas created with the framework, that span a wide range of possible speech dialogue systems. We described a service that translates speech in increments, a conversation simulation that is able to perform turn-taking, and a dialogue system that processes increments to decrease the time used to query a database.

While we focus on applications in the area of spoken dialogue, the incremental approach of this framework can be applied to other areas of research as well. For example, modules for video input and object detection can be used to reference positions of objects in the dialogue, and robotics features may be integrated so that a dialogue system can interact with its environment.

The framework is published as open source and available online at
https://www.github.com/uhlo/retico.

Acknowledgements

This work was financially supported by the German Research Foundation DFG (grant number MO 1038/23-1).

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*, pages 29–32. Association for Computational Linguistics.

ITU-T Recommendation P.805. 2007. *Subjective Evaluation of Conversational Quality*. International Telecommunication Union, Geneva.

Casey Kennington, Spyros Kousidis, and David Schlangen. 2014. Inprotks: A toolkit for incremental situated processing. *Proceedings of SIGdial 2014: Short Papers*.

Thilo Michael and Sebastian Möller. 2019. Retico: An open-source framework for modeling real-time conversations in spoken dialogue systems. In *30th Konferenz Elektronische Sprachsignalverarbeitung (ESSV)*, pages 238–245, Dresden. TUDpress.

Thilo Michael and Sebastian Möller. 2020. Simulating turn-taking in conversations with varying interactivity. In *31th Konferenz Elektronische Sprachsignalverarbeitung (ESSV)*, pages 208–215, Dresden. TUDpress.

Andrew Rafla and Casey Kennington. 2019. Incrementalizing rasa's open-source natural language understanding pipeline. *arXiv preprint arXiv:1907.05403*.

David Schlangen and Gabriel Skantze. 2009. A general, abstract model of incremental dialogue processing. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pages 710–718. Association for Computational Linguistics.

David Schlangen and Gabriel Skantze. 2011. A general, abstract model of incremental dialogue processing. *Dialogue and Discourse*, 2(1):83–111.

Ethan O Selfridge, Iker Arizmendi, Peter A Heeman, and Jason D Williams. 2011. Stability and accuracy in incremental speech recognition. In *Proceedings of the SIGDIAL 2011 Conference*, pages 110–119. Association for Computational Linguistics.

Gabriel Skantze. 2017. Towards a general, continuous model of turn-taking in spoken dialogue using lstm recurrent neural networks. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 220–230.