RETICO: AN OPEN-SOURCE FRAMEWORK FOR MODELING REAL-TIME CONVERSATIONS IN SPOKEN DIALOGUE SYSTEMS

Thilo Michael[1], *Sebastian Möller*[1,2]
[1]*Quality and Usability Lab, Technische Universität Berlin*
[2] *German Research Center for Artificial Intelligence (DFKI), Berlin, Germany*
thilo.michael@tu-berlin.de

Abstract: In this paper we present ReTiCo, a python-based programming framework that utilizes the concepts of incremental processing to create interactive spoken dialogue systems and simulations of conversational behavior. In contrast to already existing toolkits like InproTK, our framework allows for quick visual creation of complex networks and is able to save and load incremental networks for simulations, automated testing as well as analysis. It is focused on simplifying the use of incremental modules to make research on specific tasks of an incremental dialogue system (like real-time speech signal processing) and on the interaction between different incremental modules easier.
We make this framework accessible as open source so that it can be used in research on spoken dialogue systems and conversation simulation.

1 Introduction

Research in the domain of Spoken Dialogue Systems (SDS) shifted from linear processes where each component fulfilled a specific task to complex real-time systems. These modules of spoken dialogue systems (like speech recognition, natural language understanding, dialogue management, etc.) are working incrementally, that means they need to work on early hypotheses of other modules and should produce their own output as soon as possible. Because of this interconnectedness and complexity, incremental modules need to be integrated and tested in fully incremental dialogue systems. This poses a challenge for most researchers that often do not have the time and detailed knowledge to implement a complete incremental SDS to evaluate the part they are researching.

The Incremental processing ToolKit (InproTK, [1]) is an open source library that utilizes the incremental model described in [2] to enable researchers to develop real-time incremental spoken dialogue systems. This toolkit declares a unified programming interface for incremental modules and provides implementations for tasks like speech synthesis and speech recognition. However, a framework to integrate, evaluate and simulate incremental modules is missing.

To make research on incremental spoken dialogue systems and the integration of modules into such systems easier, we present ReTiCo, an incremental framework for building real-time conversation-related applications. The framework includes a simple programming interface for creating new incremental modules, as well as already implemented modules with state-of-the-art models for tasks like speech recognition, speech synthesis, natural language understanding and dialogue management. Additionally, ReTiCo provides a graphical user interface to connect incremental modules and quickly build networks. This can be useful when comparing two different versions of a module in a larger network.

The framework is written in Python, was tested on all major operating system and is available as an open source project[1].

[1]Available at `https://github.com/Uhlo/retico`

2 Related Work

The conceptual idea of incremental modules forming hypotheses and transmitting them in incremental modules was formalized the first time in [3]. InproTK [1, 4] and the subsequent InproTKs [5] for situational dialogues is based on this model.

Especially components of spoken dialogue systems like speech recognition [6] and end-of-turn prediction [7, 8] rely on incremental processing to form early hypotheses and thus need to be integrated and evaluated in an incremental framework. Dialogue Manager that try to facilitate turn-taking and backchanneling [9] need to incorporate early hypotheses from end-of-turn predictors and natural language understanding modules.

Incremental processing and integrated evaluation is not only relevant for modules of spoken dialogue systems, but also research on specialized systems like phonetically responsive spoken dialogue systems [10] and human-to-human conversation simulation [11] require an incremental environment.

3 Architecture

The architecture of ReTiCo is largely based on the conceptual model described in [2] and implemented in [4]. In this model, small increments of hypotheses formed from the current state of information are transmitted via *Incremental Units* (IU). These store references to earlier created IUs as well as references to the IUs they are grounded in. Every *Incremental Module* has a *left buffer* (input buffer) and *right buffer* (output buffer), where incoming and outgoing IUs are queued. Each incremental module has a processor that takes a number of incremental modules from the left buffer, processes them to form a new hypothesis in form of an incremental unit and puts it into its right buffer. In contrast to InproTK [4, 1], some key changes to the architecture of incremental modules and units were made that are described in the following sections.

Networks that are build with the framework can be written to a file and loaded again. This is useful for running automated evaluations or simulations with different networks and can also help with collaborating with other researchers.

3.1 Incremental Units

In this framework, incremental units (IU) are defined mostly by their payload (the increment of information they are transporting between different IUs). They contain a links to their *creator* (the incremental module that created the specific IU), the previous IU (the previous IU created by the same module) and the IU it is based on. References to these earlier IUs are only stored for a limited number of time to avoid wasting computer memory.

Specific types of IUs like `AudioIU` and `TextIU` may specify their own set of attributes that make up the payload of that IU. Because incremental units are always produced by an incremental module, the framework adds this information automatically to the IU.

Additionally to the payload, IUs may also include meta data. This meta data is not required for any IU, but may be useful for transmitting debugging information.

3.2 Incremental Modules

Incremental modules are designed to handle all concurrency that is necessary for incremental communication with other modules. They contain a left buffer and a right buffer which are embedded into a continuous loop. In most cases, an incremental module only needs to implement a method handling one incremental unit, while the framework handles the parallelization

and the passing of the incremental units. An incremental module defines one or more types of incremental units that it is able to process and one type of incremental unit that it is able to produce. That way, two independently created module may communicate with each other over previously specified incremental unit types. Incremental modules have a setup and a teardown function that is called before a network is run. This allows each module to allocate resources and setting up resources before the network is activated.

Like incremental units, incremental modules may also contain meta data. This information is saved when a network is written to file and contains data like position and size for a graphical representation of the network (see Section 5).

Besides the standard incremental module, there are three special types that are predefined: the consuming modules, producing modules and trigger modules. Producing modules are used for modules that do not process any incremental units but rather generate increments from other sources (like a microphone). In contrast to this, a consuming module only processes IUs and does not produce any (e.g a speaker module). A trigger module is a special form of producing module that does not continuously produce output, but submits a new IU whenever a trigger-function is called.

The general interface and the specification of input and output IU types allow for clean programming interface and an easy-to-use network-building interface that allows to connect modules that have matching IU types.

3.3 Event System

To allow for communication between incremental modules and to components outside of the incremental framework, incremental modules are able to register events. This is intended to be helpful for signaling certain actions that take place inside a module, for example a dialogue manager might trigger an event when the dialogue is finished. Each event is identified by a name and may include additional information about the event.

At any time it is possible to subscribe to one or all events of a specific module with a callback function that is executed as soon as a matching event triggers.

4 Predefined Modules

The most basic set of incremental modules included in the ReTiCo framework handles the recording, processing and output of audio. The `MicrophoneModule` and the `SpeakerModule` are able to access the hardware connected to the machine to record and play audio at any specified rate. The `AudioDispatcherModule` allows for granular control over when audio should be dispatched (e.g. to a speaker). An `AudioRecorderModule` saves all incoming audio to a file. This is often very useful for logging purposes.

ReTiCo uses the Google ASR and TTS services for speech recognition and speech synthesis. Additionally a MaryTTS [12] server may be used for synthesis. The Natural language understanding module uses the machine learning approach from rasaNLU [13]. For natural language generation a simple sentence selection module is implemented.

The framework implements three different types of dialogue manager, namely an n-gram dialogue manager, an agenda-based dialogue manager and a recurrent neural network dialogue manager based on rasa [13]. A turn-taking dialogue management module uses simple turn taking rules together with an end-of-turn-prediction for proper turn-taking and backchanneling.

5 Network Builder

Networks built with the ReTiCo framework can become large very quickly and connecting lots of modules in code is often confusing and unintuitive. The network builder is a graphical interface that runs in the browser. There, incremental modules can be added to a canvas, connected to each other, run, save and load

Every module is visualized as a box that contains the title of the module and parameters that were set (for example the language of the speech recognition, see Figure 1). The modules have an *enable* and *disable* button for a quick control over which modules should be active. On the corners of each module there are connection buttons for outgoing IUs (outward-facing arrows) and incoming IUs (inward-facing arrows). Clicking on one of these buttons highlights all modules that this module is able to connect to (i.e. all modules that produce or consume the same type of IUs). During execution of the network, the content of the modules switches to information about the latest incremental unit that was produced by that module (see Figure 2).

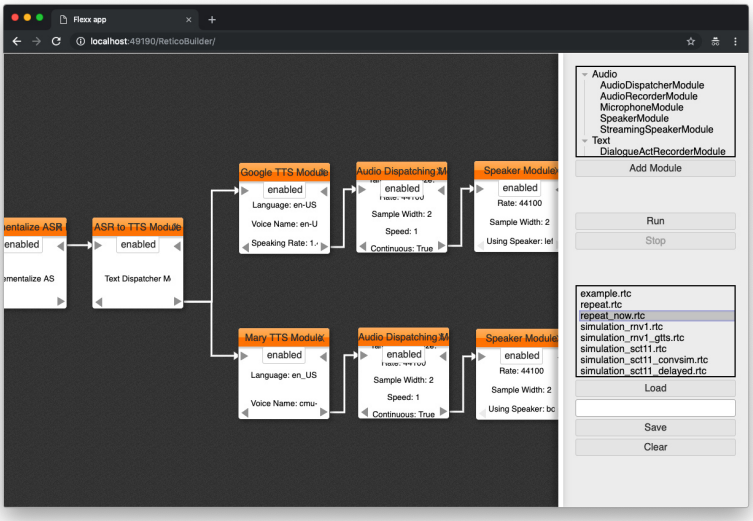

Figure 1 – A screen capture from the Network Builder web interface with an open project (center), a list of available modules (top right) and a list of already saved networks (bottom right).

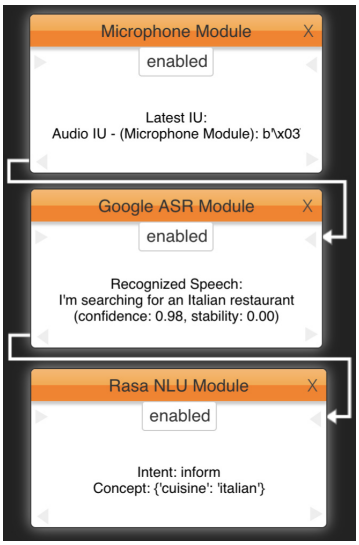

Figure 2 – Three connected modules inside the ReTiCo Network Builder.

On the top right is a tree-list-view that contains all available incremental modules. Modules like Google TTS and ASR, as well as RasaNLU only appear after the required dependencies are installed. When adding a module to the canvas, a small dialogue prompts for the parameters that are needed to initialize the module.

The Network Builder allows users to save networks to a file and load them. Networks that were created without the graphical user interface can be loaded as well (obviously, the position of the modules on the canvas is not defined in this case).

TriggerModules, that generate an incremental unit of a certain type when triggered, have a special interface in the Network Builder. During execution, these modules show an input field, where users may add a payload for the IU that is being created once the trigger-button is pressed.

6 Example Systems

With the range of already available incremental modules and the Network Builder to connect modules by placing them on a canvas, creating networks becomes easier with almost no programming effort required. In the following section we describe two example systems that we have realized with the ReTiCo framework.

6.1 Sentence Repeating

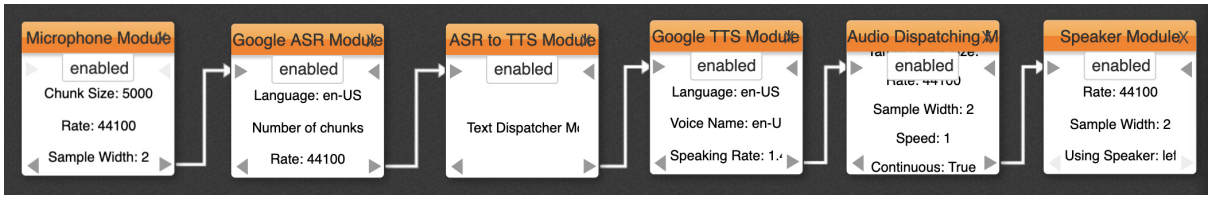

Figure 3 – A network that repeats what the user is saying by synthesizing the recognized speech.

The most simple network (other than creating a feedback loop by just connecting a microphone with a speaker) is a sentence repeating network, where the users utterance is recognized, synthesized and spoken back to the user. As shown in Figure 3, this can be realized by connecting the microphone to the speech recognition module. After that a text dispatching module is used, that collects results from the speech recognition until it detects a pause. Then, the sentence is passed to a speech synthesis module and the resulting audio is handed to a dispatching module that dispatches the audio in packets to the speaker module.

6.2 Conversation Simulation

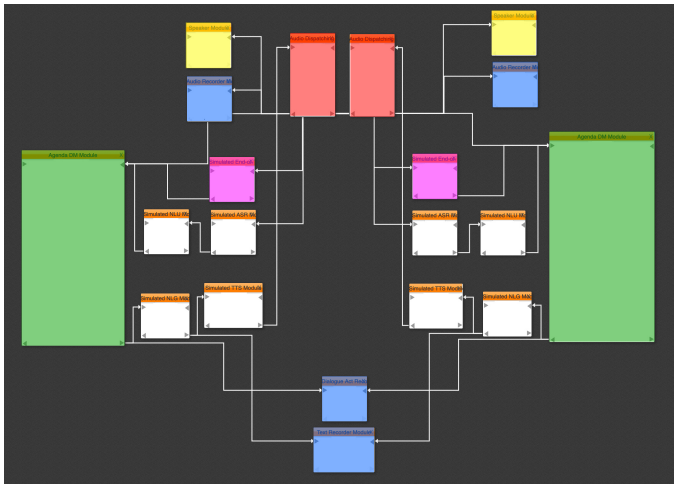

Figure 4 – A color-coded schema of a network that simulates a conversation with two virtual agents talking to each other. Blue are logging modules, green are the dialogue managers, pink the end-of-turn predictors, red the audio dispatcher modules and yellow the speaker modules.

A more complex example of a network is a conversation simulation. Figure 4 shows a color-coded schema of such a simulation. The network contains basically two spoken dialogue systems and thus the schema is mirrored. The red modules at the top of the figure are the two

audio dispatcher modules from both agents. They dispatch their audio to two speaker modules (in yellow), to the audio recorder modules (in blue), to their own dialogue manager (in green) and to the end of turn-predictor of the other agent (in pink) and finally to the speech recognition modules of the other agent.

The dialogue manager of each agent receives IUs from three different modules: from its end-of-turn predictor to estimate when the turn of the interlocutor is over, from its own NLU module to process what the interlocutor is saying and from its own audio dispatcher module to keep track of when the agent speaks. The dialogue act the dialogue manager produces contains a dispatch-flag so that early hypotheses may already be synthesized. When the dialogue manager decides to start speaking, the TTS module has a buffered output ready. The two blue modules in the bottom are logging the produced dialogue acts and natural language sentences to disk.

This incremental setup allows the two agents in this network to converse over a long period of time about the topic defined by the dialogue manager with a simple turn-taking strategy so that sometimes utterances overlap each other.

7 Discussion

In this paper we presented ReTiCo, a framework for building real-time incremental conversational systems that includes a variety of modules and offers a graphical user interface, the Network Builder to easily link together incremental modules into a network. We explained the architecture and the functionality of the user interface and outlined two example systems.

We think that this framework may be useful for researches who want to build, evaluate or integrate spoken dialogue systems, conversation simulations or any other kind of incremental speech system.

In future work we want to improve the Network Builder for better ease of use and performance and we also want to include more incremental modules (like Sphinx speech recognition) to make the framework useful for various different application areas.

Acknowledgements

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

References

[1] BAUMANN, T., O. BUSS, and D. SCHLANGEN: *Inprotk in action: Open-source software for building german-speaking incremental spoken dialogue systems*. In *Proceedings of Elektronische Sprachsignalverarbeitung (ESSV) 2010*. 2010.

[2] SCHLANGEN, D. and G. SKANTZE: *A general, abstract model of incremental dialogue processing*. *Dialogue and Discourse*, 2(1), pp. 83–111, 2011.

[3] SCHLANGEN, D. and G. SKANTZE: *A general, abstract model of incremental dialogue processing*. In *Proceedings of the 12th Conference of the European Chapter of the Association for Computational Linguistics*, pp. 710–718. Association for Computational Linguistics, 2009.

[4] BAUMANN, T. and D. SCHLANGEN: *The inprotk 2012 release*. In *NAACL-HLT Workshop on Future Directions and Needs in the Spoken Dialog Community: Tools and Data*, pp. 29–32. Association for Computational Linguistics, 2012.

[5] KENNINGTON, C., S. KOUSIDIS, and D. SCHLANGEN: *Inprotks: A toolkit for incremental situated processing*. *Proceedings of SIGdial 2014: Short Papers*, 2014.

[6] SELFRIDGE, E. O., I. ARIZMENDI, P. A. HEEMAN, and J. D. WILLIAMS: *Stability and accuracy in incremental speech recognition*. In *Proceedings of the SIGDIAL 2011 Conference*, pp. 110–119. Association for Computational Linguistics, 2011.

[7] HARA, K., K. INOUE, K. TAKANASHI, and T. KAWAHARA: *Prediction of turn-taking using multitask learning with prediction of backchannels and fillers*. *Proc. Interspeech 2018*, pp. 991–995, 2018.

[8] MAIER, A., J. HOUGH, and D. SCHLANGEN: *Towards deep end-of-turn prediction for situated spoken dialogue systems*. In *Proceedings of INTERSPEECH 2017*. 2017.

[9] TER MAAT, M., K. P. TRUONG, and D. HEYLEN: *How agents' turn-taking strategies influence impressions and response behaviors*. *Presence: Teleoperators and Virtual Environments*, 20(5), pp. 412–430, 2011.

[10] RAVEH, E., I. STEINER, and B. MÖBIUS: *A computational model for phonetically responsive spoken dialogue systems*. *Proc. Interspeech 2017*, pp. 884–888, 2017.

[11] MICHAEL, T. and M. SEBASTIAN: *Simulating human-to-human conversations for the prediction of conversational quality*. *Fortschritte der Akustik-DAGA*, 2018.

[12] SCHRÖDER, M. and J. TROUVAIN: *The german text-to-speech synthesis system mary: A tool for research, development and teaching*. *International Journal of Speech Technology*, 6(4), pp. 365–377, 2003.

[13] BOCKLISCH, T., J. FAULKER, N. PAWLOWSKI, and A. NICHOL: *Rasa: Open source language understanding and dialogue management*. *arXiv preprint arXiv:1712.05181*, 2017.