

ScriptBoard: Designing modern spoken dialogue systems through visual programming

Divesh Lala, Mikey Elmers, Koji Inoue, Zi Haur Pang, Keiko Ochi, Tatsuya Kawahara
Kyoto University Graduate School of Informatics

Japan

lala@sap.ist.kyoto-u.ac.jp

Abstract

Implementation of spoken dialogue systems can be time-consuming, in particular for people who are not familiar with managing dialogue states and turn-taking in real-time. A GUI-based system where the user can quickly understand the dialogue flow allows rapid prototyping of experimental and real-world systems. In this demonstration we present ScriptBoard, a tool for creating dialogue scenarios which is independent of any specific robot platform. ScriptBoard has been designed with multi-party scenarios in mind and makes use of large language models to both generate dialogue and make decisions about the dialogue flow. This program promotes both flexibility and reproducibility in spoken dialogue research and provides everyone the opportunity to design and test their own dialogue scenarios.

1 Introduction

In spoken dialogue system (SDS) research the methodology or tools used to create interactions is often not fully described. A common approach is to use finite state machines, but these may grow unwieldy with more complex interactions because of the large number of states. Behavior trees are another approach that is used for AI in video games (Colledanchise and Ögren, 2018), but programming these may require a steeper learning curve.

Researchers who are unfamiliar with implementing SDSs often need time to learn how to manage aspects such as turn-taking and handling dialogue states in real-time systems, where human and robot turns are not neatly separated and system interruptions can be frequent. With no existing standards for the design of SDSs, researchers need to create even simple interactions from scratch. Furthermore, it is often difficult for these interactions to be reused and modified by others if they are written exclusively using code with no graphical interface.

We propose that due to the above issues a GUI

should be used to assist in quickly designing replicable spoken dialogue interactions. There are several visual programming approaches which have been used in the literature, but these were developed before the rise of two research fields in SDSs - large language models (LLMs) and multi-party interactions. An updated approach would need to accommodate these aspects in the system. LLMs drastically reduce the number of dialogue generation states needed in a visual programming interface since one LLM node can handle both natural language understanding (NLU) and response generation. Therefore the graphical design of complex interactions becomes more viable for novice users.

In this demonstration we present ScriptBoard (**Script Builder Offering Assistance with Robot Dialogue**), a visual programming system which can be used to quickly develop SDSs. ScriptBoard has the ability to handle multiple human participants, integrates prompt-based LLMs into the dialogue flow, and handles spoken dialogue features such as silence and barge-in. Our program allows researchers to create spoken dialogue scenarios for both real-world implementation and experiments.

The system is written in Python using the PyQt package for the graphical interface. In this work we define the “user” as a person who designs the dialogue scenario and the “participant” as the person who is actually involved in the interaction.

2 Related Work

Similar visual systems to manage robot interactions have been implemented in previous works (Nakano and Komatani, 2024; Groß et al., 2023; Michael, 2020; Koller et al., 2018; Lison and Kennington, 2016; Glas et al., 2016; Pot et al., 2009). These often make use of a state-machine design in which the states of an interaction are set by the user and tracked during the interaction. State transitions in spoken dialogue systems are often triggered by au-

omatic speech recognition (ASR) results, however this may not be appropriate for scenarios such as conversation where deciding when to speak (i.e. turn-taking) is crucial for smooth interactions.

Another issue is systems which are tightly coupled with a specific robot platform. For example, IrisTK is used to design interactions for Furhat (Skantze and Al Moubayed, 2012) and Choregraphe (Pot et al., 2009) is used for the NAO robot. It would be preferable if users were able to reproduce interactions with different robots, particularly for comparative evaluation. Existing systems are also usually made with the assumption of one-to-one interaction. IrisTK is made for multi-party interaction, however this is based on XML code and may be somewhat difficult for novice users. This issue of usability is also critical. Some systems such as Interaction Composer (Glas et al., 2016) and DialogOS (Koller et al., 2018) focus on making the interface understandable to novice users while more powerful systems such as RISE (Groß et al., 2023) are targeted towards researchers and may require more expert knowledge.

Recent technological advances mean that even novice users can use LLMs to generate dialogue without needing any low level programming as they can use prompt-based inputs to direct the system. A tool which could assist users to design SDS scenarios in this way would be ideal for rapid prototyping and testing. Frameworks such as Retico (Michael, 2020) and DialBB (Nakano and Komatani, 2024) integrate LLMs into their systems although they are less focused on visual programming. Scriptboard allows users to do everything within the GUI.

3 System Architecture

ScriptBoard is not a spoken dialogue system itself, but communicates with an external controller using TCP/IP messaging. The external controller handles speech recognition input from the user and turn-taking and sends this information to ScriptBoard. This information is then processed using the scenario created in the GUI by the user of ScriptBoard to decide the robot’s behavior. ScriptBoard messages are then parsed by the controller and used for behavior generation.

Natural language processing and dialogue generation can be done through simple keyword comparators, but ScriptBoard allows users to use LLMs in their scenarios. This approach simplifies the creation of interaction scenarios because the user

can hand off complex tasks to the LLM rather than handcrafting individual dialogue states for the robot.

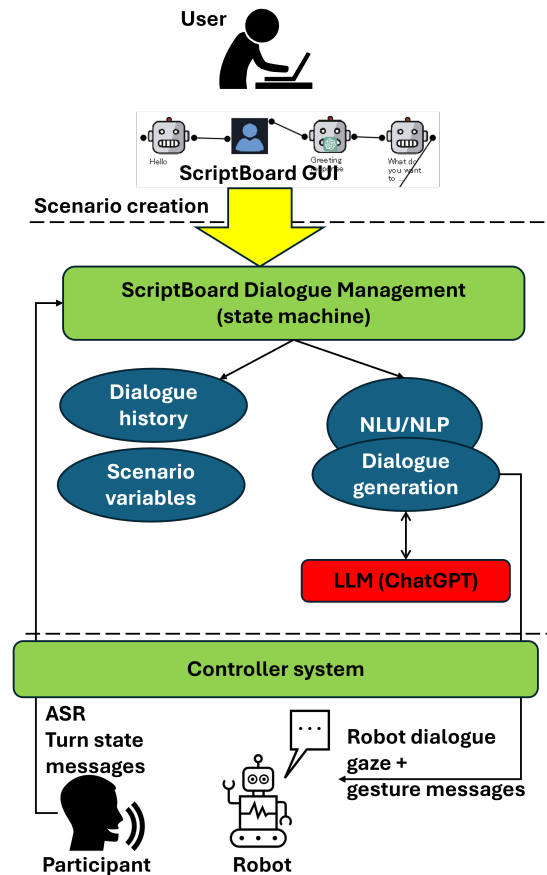


Figure 1: ScriptBoard architecture. Communication with the external controller is done through an environment-independent messaging system.

Figure 1 shows this general architecture of ScriptBoard. Communication with the controller is done only through messaging, meaning it can interface with any compatible external system and is independent of any robot or agent, speech recognition or text-to-speech (TTS) system.

4 User Interface

The user interface of ScriptBoard is based on the paradigm of dialogue states which has been used in previous visual programming systems (Glas et al., 2016). It uses a drag-and-drop mechanism in which the user can place states (also known as nodes) on the canvas. For a node, the leftmost connector represents an inbound connection to the node, while connectors on the right represent transitions out of the node. Users connect these by dragging lines to other nodes to visually represent the dialogue flow.

We use Figure 2 as a reference interaction for

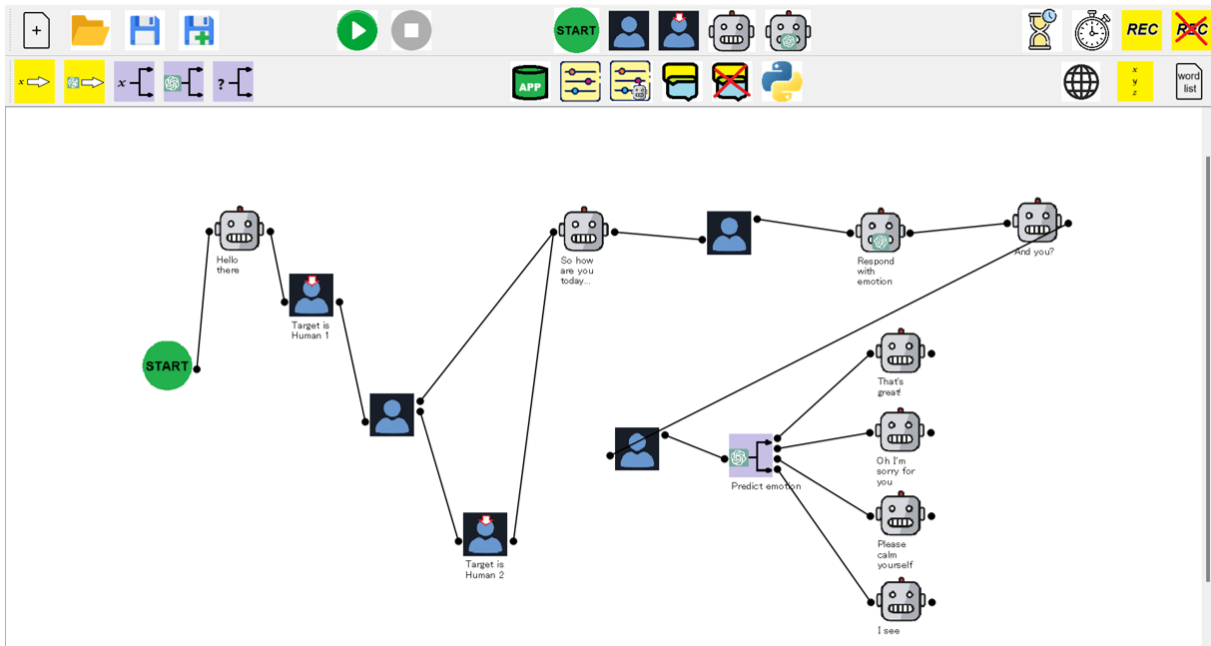







Figure 2: The GUI of ScriptBoard showing a simple interaction in a multi-party setting where the system greets each participant one at a time using LLM outputs.

this work. The toolbar at the top of the GUI contains various nodes for the scenario. We create this interaction by connecting the following nodes which we will describe in more detail.

-  Human turn node which processes speech from the participant
-  Sets the target participant
-  Generates dialogue for the robot (handcrafted)
-  Generates dialogue for the robot using a ChatGPT response
-  Controls dialogue flow based on output from ChatGPT


In the interaction in Figure 2, the system detects who responds first and then asks how they are. The first dialogue is automatically generated by ChatGPT. The system then asks the next participant the same question, but in this case the ChatGPT output is simply the predicted emotion of the participant’s preceding utterance. This output (happy, sad, angry or other) is then used as a condition to the corresponding handcrafted dialogue. We now describe the details of these nodes.


4.1 Setting the environment

Before a scenario can be executed the user must set the environment of the interaction by specifying the number of participants and their user identification numbers. Every ASR result received must be associated with a corresponding ID. One participant may also be assigned as the “target” participant and this can be changed during the interaction.

This feature is necessary for handling multi-party interaction. ScriptBoard users can decide which participant(s) the robot should listen to and choose dialogue flows depending on who is speaking. It allows users to set roles related to each participant in advance and track them over the course of the interaction.

In addition, the user may set a number of variables to track during the interaction which can be modified. These variables are used as conditions to change the dialogue flow and are strictly enforced as either string, integer, float or boolean types.

 Click to set information about the participants in the interaction

 Click to set variables tracked during the interaction

4.2 Human turn nodes

ScriptBoard is driven by conversational turn states. There are four basic states which occur in a basic

cycle: *human turn, offer to robot, robot turn and offer to human*. The decision on the turn-taking state is made by the external controller, which sends turn update messages to ScriptBoard.

The human turn node is entered during the human turn state, and waits for user utterances. The node adds any incoming ASR result to the dialogue history of the relevant participant. ASR results need to be tagged with the corresponding ID number so that this dialogue history is accurate.

The system waits for an ASR result and/or the end of a participant’s turn, which is notified by the external controller. Users can set conditions for either, such as whether the string contains a word, starts with or ends with a certain string(s) or is over a certain length. Conditions for a participant’s utterance are checked as soon as an ASR result is received, while conditions for a participant’s turn are checked as soon as the turn state changes to *offer to robot*, or in the case of multiple participants, the *human turn* state changes to a different participant. The user can also define whether a condition is for a particular participant, or a defined target or non-target participant or both.

Multiple conditions can be set for a human node which are checked sequentially. Each of these will generate a connector allowing them to be connected to other nodes. Figure 3 shows the two conditions used for the first (leftmost) human turn node in Figure 2. The first condition is if the target participant’s turn is more than 10 words. The next condition consists of two sub-conditions - the non-target’s utterance contains the word “hello” and is more than 10 words. Figure 2 shows that this second condition branches to a node which sets the participant with ID 2 as the new target.

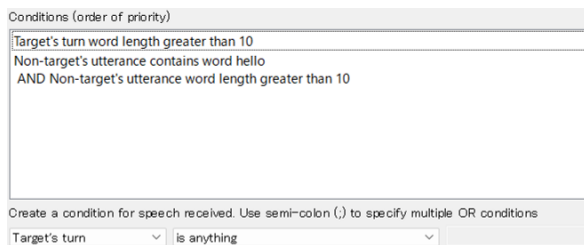


Figure 3: Human state window displaying the two conditions in Figure 2’s scenario.

4.3 Robot nodes and LLM integration

Robot nodes specify dialogue to be generated by the system. The user can handcraft the dialogue themselves and specify speech tags and gaze which

send extra information which may be used by the external robot controller. The interaction in Figure 2, shows examples of this node when the robot says “Hello there” and “How are you today?”.

The approach described above of checking and comparing keywords and then manually generating a response is somewhat naive and impractical for many situations. Nowadays state-of-the-art LLMs are able to do both NLU and response generation tasks which greatly reduces the amount of effort needed to build an effective dialogue system. ScriptBoard integrates LLMs directly into the GUI through the use of a robot GPT node which uses the ChatGPT API.

With this node the user may opt to use ChatGPT to generate the response using prompts. ScriptBoard must be connected to an external ChatGPT program which can use the API. This program is included in the ScriptBoard package. The user types in their prompt directly and ScriptBoard will send the prompt and generate the response received from ChatGPT. Dialogue history can also be appended to augment the prompt. The user can also define how many turns and which participant’s dialogue should be used from the dialogue history. In Figure 2, the robot GPT node labeled “Respond with emotion” will generate a ChatGPT response. It uses the prompt “*Generate an empathetic response to the following utterance*” and specifies the most recent turn of the target participant.

The robot node will wait until the robot has said its utterance (either handcrafted or generated) before exiting to the next node. This notification is received by a message from the external controller. Therefore it is necessary for the controller to know exactly when the robot has stopped its speech.

4.4 Dialogue flow management

To manage dialogue for more complex interactions, the user may wish to adapt the dialogue flow depending on conditions which are not related to participants’ ASR results. ScriptBoard allows this control depending on variables or LLM output.

The variables specified in the initial setup of the interaction can be used to check for conditions and control dialogue flow. ScriptBoard contains a node which can be used to directly set the value of a variable during an interaction. For example, an *age* variable might be used to store the participant’s age after being prompted by the system. Another node is then used to control the dialogue flow depending on this value. This node can then be used to

produce different dialogue depending on *age*.

A similar method can be used to set variables based on results from LLMs. The user can specify prompts to generate a value rather than dialogue and store this in a variable. Assume that there is a variable named *emotion* to store the emotion of the participant. The user firstly chooses the node which stores the ChatGPT result in *emotion*. They can then use a prompt such as “*Output the sentiment of the following utterance. The possible sentiments are happy, sad, or angry*”. When this node is reached, the output of ChatGPT is stored in *emotion* and then the variable decision node can be used to control the flow depending on the value.

Another method is available where the output of ChatGPT is directly used without needing a variable. In this case the user can specify prompting as usual, but they can also control the dialogue flow depending on ChatGPT’s output. Figure 4 shows an example of this when the robot asks the second participant how they are. In this case, once the prompt is processed, the user specifies which conditions should be checked against ChatGPT’s output, expecting either “happy”, “sad” or “angry”. Four conditions are created (one is an “else” condition) and these can be connected to other nodes to control dialogue flow.



Figure 4: Robot GPT window showing the specified conditions for *Predict Emotion*.

5 Additional Features

For more complex SDSs, ScriptBoard has several other unique features for helping users customize their own scenarios.

5.1 Silence and barge-in

The system also contains features which handle phenomena that are specific to spoken dialogue systems. The first of these is silence. Silence messages are sent from the external controller. In the human turn state, the user can specify a condition which is triggered on silence for a specified time period. Use cases for this condition include prompting the user to speak or to end the interaction.

The system also has functionality to handle barge-in, the interruption of a system utterance

from the user. This is again triggered from the external controller which sends a message whenever barge-in is detected. Note that the actual barge-in model and interruption of the system’s TTS is handled by the external controller, not by ScriptBoard. Our system simply allows the user to define the dialogue flow which occurs when barge-in occurs.

5.2 Extendability

Although we use ChatGPT as the default LLM in this work, ScriptBoard also allows researchers to integrate their own models into the dialogue scenario through TCP connections. The input utterances for a model can be specified and resulting dialogue used in the interaction. This allows for quick prototyping and evaluation of SDSs.

In addition to LLMs, other types of conversational models can be triggered by ScriptBoard, by customizing message protocols. In our work we have successfully executed backchannelling and laughter models for a robot through this process. ScriptBoard also allows users to use a node to launch their own customized Python functions, making it a useful tool for proficient programmers.

6 System Usage

Once the user has created their dialogue scenario and it is connected to the external controller, they can simply push the “Play” button in the top toolbar to start the interaction. The user can view the dialogue history and the variables in the scenario through a monitoring window. The GUI also focuses the viewpoint on the current node to let the user know exactly where they are in the scenario, allowing them to visually track its progress. This is useful for debugging the logic of the scenario or identifying areas of improvement.

The scenario should run autonomously until there are no more connected nodes, but it is possible for an interaction to run in a loop. ScriptBoard can be for a diverse number of systems and robots. We describe one such implementation in a multi-party setting.

6.1 Multi-party attentive listening

This scenario was conducted in a public exhibition using a CommU robot¹. It required two participants and a robot who would engage in an attentive listening dialogue. A microphone array separated each participant’s voice so speech recognition

¹https://resou.osaka-u.ac.jp/en/research/2015/20150120_2

could be performed on multiple channels simultaneously (Ishikawa et al., 2024). There were three parts to this scenario as shown in Figure 5:

1. Introduction in which robot greets both participants and explains attentive listening.
2. Attentive listening dialogue where robot listens to each participant one at a time for one minute.
3. A tongue twister game to demonstrate how the robot can listen to two people at once. Participants said a designated tongue twister together and the robot played back their separated audio then announced a winner.

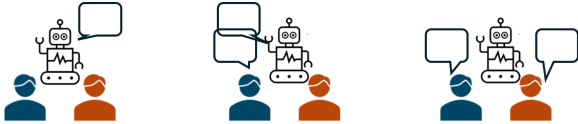


Figure 5: Overview of attentive listening scenario. The robot first acts as an explainer then as an attentive listener before participants play a simultaneous speaking game.

The first part required that the robot act as an explainer by gazing between both participants and acknowledging them as part of the scenario. ScriptBoard was used to receive confirmation utterances from both participants. It could also set each participant as a gaze target by simply including this information with the robot’s utterance.

For the second part we used ScriptBoard to have the robot act as an attentive listener. Responses were generated using a prompt which generated attentive listening style responses. We used a timer which would end the dialogue after a set amount of time. A backchannel model was called so that it would run during attentive listening.

For the tongue twister game the robot had to receive both participants’ ASR simultaneously, use it to decide the winner of the game and also play back their separated audio channel. ScriptBoard processes ASR from both participants and this was made simple through designing the scenario’s environment. Playback and deciding the winner could also be achieved through customized functions.

6.2 Other usages

The above scenario describes a mostly chatting-based interaction. However ScriptBoard is not limited to these types of interactions and should also

be able to reliably handle task-based interactions by using LLMs and an appropriate prompt. Furthermore, because ScriptBoard is driven by turn-taking, these are abstracted from the type of interaction and so can be used in any spoken dialogue scenario.

As an example of the above, we have also used ScriptBoard to control a job interview task in English with an android robot (Pang et al., 2024). Although this task requires a different style of talk and a slightly longer time between turns, ScriptBoard was able to manage this interaction in a mostly linear dialogue flow.

ScriptBoard can even be used without needing any robot (such as a voice assistant) since it only outputs messages containing a response. Any connected system can receive this message and decide how it should be executed.

6.3 Reproducibility

SDS literature often describes human-robot conversational systems used in experiments and fieldwork, but it is difficult for others to use the system or create the same scenario without the available source code. ScriptBoard scenarios are saved as a JSON file to make them reproducible. Another ScriptBoard user can then easily load and test this scenario in their own environment.

Furthermore, scenarios written in ScriptBoard can be easily applied to other robots. In the job interview system described above, we used two different robots (Pang et al., 2025) each running the same ScriptBoard scenario, demonstrating how the same dialogue logic can be used in different robots. This would allow researchers to share their dialogue system and allow others to test in their own particular robot.

7 Conclusion

We demonstrate the ScriptBoard system, which we use to design and implement spoken dialogue systems using visual programming. ScriptBoard uses turn states as a basis for dialogue management and incorporates recent advances in LLM technology and multi-party scenarios. It is independent of any agent or robot and we have used it in different types of scenarios to demonstrate its capabilities.

Acknowledgments

This work was supported by JST Moonshot R&D Goal 1 Avatar Symbiotic Society Project (JP-MJPS2011).

References

- Michele Colledanchise and Petter Ögren. 2018. *Behavior trees in robotics and AI: An introduction*. CRC Press.
- Dylan F. Glas, Takayuki Kanda, and Hiroshi Ishiguro. 2016. Human-robot interaction design using interaction composer: Eight years of lessons learned. In *The Eleventh ACM/IEEE International Conference on Human Robot Interaction, HRI '16*, page 303–310. IEEE Press.
- André Groß, Christian Schütze, Mara Brandt, Britta Wrede, and Birte Richter. 2023. [Rise: an open-source architecture for interdisciplinary and reproducible human–robot interaction research](#). *Frontiers in Robotics and AI*, 10.
- Yuto Ishikawa, Kohei Konaka, Tomohiko Nakamura, Norihiro Takamune, and Hiroshi Saruwatari. 2024. [Real-time speech extraction using spatially regularized independent low-rank matrix analysis and rank-constrained spatial covariance matrix estimation](#). In *2024 IEEE International Conference on Acoustics, Speech, and Signal Processing Workshops (ICASSPW)*, pages 730–734.
- Alexander Koller, Timo Baumann, and Arne Köhn. 2018. Dialogos: Simple and extensible dialogue modeling. In *Interspeech 2018*, pages 167–168.
- Pierre Lison and Casey Kennington. 2016. [OpenDial: A toolkit for developing spoken dialogue systems with probabilistic rules](#). In *Proceedings of ACL-2016 System Demonstrations*, pages 67–72, Berlin, Germany. Association for Computational Linguistics.
- Thilo Michael. 2020. [Retico: An incremental framework for spoken dialogue systems](#). In *Proceedings of the 21th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 49–52, 1st virtual meeting. Association for Computational Linguistics.
- Mikio Nakano and Kazunori Komatani. 2024. [DialBB: A dialogue system development framework as an educational material](#). In *Proceedings of the 25th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 664–668, Kyoto, Japan. Association for Computational Linguistics.
- Zi Haur Pang, Yahui Fu, Divesh Lala, Mikey Elmers, Koji Inoue, and Tatsuya Kawahara. 2024. Human-like embodied AI interviewer: Employing android ERICA in real international conference. In *COLING*. (to appear).
- Zi Haur Pang, Yahui Fu, Divesh Lala, Mikey Elmers, Koji Inoue, and Tatsuya Kawahara. 2025. [Does the appearance of autonomous conversational robots affect user spoken behaviors in real-world conference interactions?](#) *Preprint*, arXiv:2503.13625.
- E. Pot, J. Monceaux, R. Gelin, and B. Maisonnier. 2009. [Choregraphe: a graphical tool for humanoid robot programming](#). In *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, pages 46–51.
- Gabriel Skantze and Samer Al Moubayed. 2012. [Iristk: a statechart-based toolkit for multi-party face-to-face interaction](#). In *Proceedings of the 14th ACM International Conference on Multimodal Interaction, ICMI '12*, page 69–76, New York, NY, USA. Association for Computing Machinery.