ABSTRACT
We present a Social RoBoT BehavioR AuthOring (SOBOR) language that enables rapid prototyping of interactive behaviors via declarative specification. SOBOR supports trigger-action programming like and imperative scripting like specifications that enable non-roboticist programmers to express complex interactive behaviors in a concise and declarative syntax. To run the specified behaviors on a robot in a language-agnostic way, the SOBOR compiler compiles SOBOR programs into functional reactive programs that can be implemented with a cross-platform reactive programming library. We present two example SOBOR programs, demonstrating its richness in expressing both progressive and reactive interactive behaviors.

KEYWORDS
ACM Reference Format:

1 INTRODUCTION
Social robots are becoming increasingly ubiquitous across domains including entertainment, education, social-emotional learning, and mental health support, among others. Programming social robots to be robust, effective, and engaging for every unique use case and environment remains a bottleneck given the complex multi-modal, interactive nature of desired robot behaviors.

Research on end-user programming of social robots aims to address this problem by simplifying the programming process to let end-users to program robots on their own [2, 5, 6, 8, 9, 11, 14]. These simplifications often come at the cost of expressivity, i.e., robot behaviors obtainable using simplified languages are not as rich as ones created using general-purpose languages by robotics expert programmers. In the industry, personal robot companies took a similar approach, e.g., by providing software development kit (SDK) and application programming interfaces (API) to their customers so they can develop robot applications by themselves [1, 12, 16]. However, programming robots was not a trivial task for customers. Even if they could develop robot applications by themselves, having to re-program the application whenever they wanted to consider a different robot platform made it too expensive to invest their time and resources in developing robot applications.

In this paper, we present a domain-specific language for Social RoBoT BehavioR AuthOring (SOBOR). SOBOR simplifies programming interactive behaviors involving multi-modal asynchronous inputs and outputs by first defining two input types—event and state—and two output types—action and controller—and second providing ways to map inputs and outputs, e.g., in the same or different types. Using SOBOR, programmers can declaratively specify interactive behaviors using both trigger-action programming and imperative-scripting like syntax. The SOBOR compiler compiles SOBOR programs into functional reactive programs that can be implemented using a cross-platform reactive programming library like ReactiveX\footnote{https://reactivex.io/}. In contrast to many existing social robot programming systems developed by researchers targeting non-experts, SOBOR treats behavior authoring as declarative specification targeting programmers who have a limited budget on the behavior authoring task. We present two example SOBOR programs representing progressive and reactive interactive behaviors of social robots to demonstrate SOBOR’s expressivity and ease-of-use.

2 RELATED WORK
Research work on end-user programming of social robots often adopts visual programming approach to simplify the complexity of the programming process, e.g., by exposing a flowchart programming interface [2, 8, 9, 14] or block programming interface [5, 6, 13]. Perhaps a more closely related approach uses a textual programming approach [11]. Our work is similar in the spirit of simplifying social robot programming, however, we consider programmers who do not want to spend much time as the target users.

Our work is also related to SDKs and APIs provided by personal robot companies with their robot platform. These SDKs and APIs support event-driven programming using the event loop provided by JavaScript runtime environments [12], Python standard library [1], or a custom library [16]. While providing SDKs and APIs may be the most general approach, we believe such generality makes them too difficult or tedious to use, even for programmers who have a limited time budget. SOBOR is a domain-specific language that aims to balance the expressivity and ease-of-use trade-off.

Programmers often use the provided APIs to create interactive behaviors in abstract representations such as finite-state machine [3], behavior tree [7], or petri-nets [4]. These abstract representations have natural visualization (e.g., flowchart for finite-state machine) on which visual interface-based systems mentioned earlier are built. SOBOR is a domain-specific language built on top of a functional reactive programming language. It could be extended to...
At a high level, SOBORO programs can be considered as a set of value can be evaluated and accessed at any given time. We also a control signal for an actuator.

### 3.1 Input and Output Types

At a high level, SOBORO programs can be considered as a set of trigger-action rules (i.e., functions) where both triggers (i.e., inputs) and actions (i.e., outputs) are data streams. We identify two distinct types of inputs: events and states. An event is the occurrence of some change at a specific point in time and a state is a condition whose value can be evaluated and accessed at any given time. We also identify two distinct types of outputs: actions and controllers. An action is an instantaneous signal for starting a movement that eventually comes to an end and a controller are conditions representing a control signal for an actuator.

### 3.2 Syntax

Figure 1 defines the syntax of SOBORO. The \( \langle \text{behavior} \rangle \) definition describes an interactive behavior as a list of \( \langle \text{rule} \rangle \)-s, defining mappings between events and actions via \( \langle \text{when} \rangle \)-expr and states and controllers via \( \langle \text{while} \rangle \)-expr. The \( \langle \text{event} \rangle \)-expr defines the event composition expression and the \( \langle \text{action} \rangle \)-expr defines the action composition expression; they let programmers express custom trigger events and complex activation signals, respectively. Similarly, the \( \langle \text{state} \rangle \)-expr defines the state composition expression and the \( \langle \text{controller} \rangle \)-expr defines the controller composition expression. SOBORO requires programmers to specify the input and output types explicitly. For example, combining event and state or action and controller is only allowed in specific ways, i.e., via \( \langle \text{state} \rangle \)-expr involving and and \( \langle \text{action} \rangle \)-expr involving and, respectively. The term-‘repeatedly’ \( \langle \text{action} \text{−} \text{expr} \rangle \)-can be used to convert the action output type to the controller output type.

To make programming sequential action execution behaviors easy, SOBORO supports chaining multiple action expressions using the ‘THEN’ keyword which triggers the specified activation signal only when the last action triggered in the chained \( \langle \text{when} \rangle \)-expr has finished successfully. The \( \langle \text{when} \rangle \)-expr involving ‘THEN’ is a convenient syntax that can be expressed with multiple \( \langle \text{when} \rangle \)-expr-s. SOBORO also allow programmers to control the number of times to respond to the trigger events using the keywords like ‘WHEN’ and ‘WHenever’; using ‘WHEN’ makes the rule to respond to the first occurrence of the trigger event where using ‘WHenever’ makes the rule to respond to all trigger event occurrences.

### 3.3 The SOBORO Compiler

The SOBORO compiler ingests a SOBORO program and outputs a functional reactive program. Specifically, the compiler parses a SOBORO program to produce an abstract syntax tree (Figure 2a,b) and then interprets the parsed tree to produce an executable program (Figure 2c).

Figure 3 shows a snippet of the compiler implementation. Using the syntax defined in Section 3.2, one can implement the parser function with an off-the-shelve parser generator like PEG.js². The interp function implements the semantics of SOBORO syntax. For example, the function interprets the parts of the syntax tree corresponding to \( \langle \text{when} \rangle \)-expr into a set of statements that apply reactive programming operators to input sources, i.e., event and state data streams (Figure 2c L8-L20). The sub-trees corresponding to \( \langle \text{event} \rangle \)-expr and \( \langle \text{when} \rangle \)-expr are interpreted as statements that create trigger event data streams from the input event and state data streams (Figure 3 L22) and map the trigger data streams to emit action values and merge the mapped data streams with the outgoing action data streams (Figure 3 L25-L31), respectively.

The SOBORO compiler is not tied to a particular data format, programming language, or reactive programming library. We represented abstract syntax trees is in JSON (JavaScript Object Notation) and compiled programs in JavaScript using RxJS³ and implemented the compiler in JavaScript for explanatory purposes only. Other

---

²https://pegjs.org/
³https://rxjs.org/
We present two example SOBORO programs written for the ide-
4
alized social robot (e.g., consisted of a tablet “face”, similar to the
one introduced in [6]) that is capable of detecting voice commands,
tracking a face, speaking, and making eye movements. We assume
the following input events
• Ready indicates that the robot is ready.

data formats like YAML (Yet Another Markup Language), program-
ning language like Python, and reactive programming library like
RxPy\(^4\) can be used to implement the compiler.

4 EXAMPLE INTERACTIVE BEHAVIORS

We present two example SOBORO programs written for the ide-
alized social robot (e.g., consisted of a tablet “face”, similar to the
one introduced in [6]) that is capable of detecting voice commands,
tracking a face, speaking, and making eye movements. We assume
the following input events
• HumanSpeech is an output of the speech recognizer.
• Time represents the time as a discrete event.

and input states
• HumanFace is a visibility state of the human face.

are available to the robot. As for the outputs, we assume the fol-
lowing actions
• Say causes the robot to say a phrase.
• PlaySound starts playing the specified sound file.

and controllers
• SetImageTo displays the specified image.
• SetEyePosX: moves the eyes to the specified location in x-
axis.
• SetEyePosY: moves the eyes to the specified location in y-
axis.

are available to the robot.

\(^4\)https://rxpy.readthedocs.io/en/latest/get_started.html

Figure 2: An example interactive behavior in the representa-
tions involved in the SOBORO compiler.

Figure 3: A snippet of an example compiler implementation.
4.1 Interactive Storytelling

The first example implements the storytelling behavior that can wait for human inputs such as verbal response and attention (simplified via the visible human face state, `HumaFace` is “visible”) to make progress in narrating the story. When the human is engaged (i.e., “visible”), the robot looks at the human to establish the mutual gaze. The program is mainly sequential yet responsive to the human inputs, demonstrating the SOBORO’s expressivity.

4.2 Meditation Guide

```plaintext
1 // Scheduled meditation guide
2 WHENEVER Time is "8:00am"
3 PlaySound "morning_meditation_sound.mp3" or PlaySound
   "morning_meditation_instructions.mp3"
4 WHENEVER Time is "4:00pm"
5 PlaySound "afternoon_meditation_sound.mp3" or PlaySound
   "morning_meditation_instructions.mp3"
6 // On-demand meditation guide
7 WHENEVER HumanSpeech is "play background music"
8 repeatedly PlaySound "background_meditation_sound.mp3"
9 WHENEVER HumanSpeech is "stop"
10 StopPlaySound
```

The second example implements a mainly reactive behavior. It plays two different mediation sounds or instructions at the scheduled time. The human can also start another meditation sound on-demand or stop any sound using a verbal command.

5 FUTURE WORK

SOBORO is in a preliminary state and multiple improvements can be made. To make SOBORO more practical, typical language features like variables and composition patterns should be supported, which we believe can be done by leveraging existing solutions used in the domain-specific languages for creating chatbots. As SOBORO becomes more complex, it may difficult to extend the current natural language-like format. We plan to investigate the possibility of applying one of the functional reactive program verification techniques recently proposed by programming language researchers.

6 CONCLUSION

In this paper, we presented SOBORO, a domain-specific language for authoring interactive robot behaviors targeting programmers with a limited time budget. The syntax of SOBORO allows users to express complex interactive behaviors consisting of sequential and reactive behaviors with ease. The SOBORO compiler interprets SOBORO programs as functional reactive programs and supports outputting the compiled program in a language-agnostic way.

We believe our proposed approach of targeting non-roboticist programmers to author interactive robot behaviors via declarative specification is an effective and interesting take on end-user programming of social robots research. We believe this paper opens up new and exciting research directions such as developer tools such as verifiers, high-level interaction grammar design, and further applications.

REFERENCES