SOBORO: A Social Robot Behavior Authoring Language

Michael Jae-Yoon Chung and Maya Cakmak
mjyc,mcakmak@cs.washington.edu
Computer Science & Engineering
Seattle, Washington, USA

ABSTRACT

We present a SOcial RoBOt BehavioR AuthOring (SOBORO) language that enables rapid prototyping of interactive behaviors via declarative specification. SOBORO 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 SOBORO compiler compiles SOBORO programs into functional reactive programs that can be implemented with a cross-platform reactive programming library. We presents two example SOBORO programs, demonstrating its richness in expressing both progressive and reactive interactive behaviors.

KEYWORDS

ACM Reference Format:
Michael Jae-Yoon Chung and Maya Cakmak. 2022. SOBORO: A Social Robot Behavior Authoring Language. In *Proceedings of International Conference on Human-Robot Interaction (HRI).* ACM/IEEE, Online

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 the customers 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 (SOBORO). SOBORO 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 SOBORO, programmers can declaratively specify interactive behaviors using both trigger-action programming and imperative-scripting like syntax. The SOBORO compiler compiles SOBORO programs into functional reactive programs that can be implemented using a cross-platform reactive programming library like ReactiveX[1]. In contrast to many existing social robot programming systems developed by researchers targeting non-experts, SOBORO treats behavior authoring as *declarative specification* targeting *programmers* who have a limited budget on the behavior authoring task. We present two example SOBORO programs representing progressive and reactive interactive behaviors of social robots to demonstrate SOBORO'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. SOBORO 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 some visual interface-based systems mentioned earlier are built. SOBORO is a domain-specific language built on top of a functional reactive programming language. It could be extended to

[1]https://reactivex.io/

HRI, March 07–10, 2022,

⟨*behavior*⟩ ::= '[' ⟨*rule*⟩, ⟨*rule*⟩, ... ']'

⟨*rule*⟩ ::= ⟨*when-expr*⟩ | ⟨*while-expr*⟩

⟨*when-expr*⟩ ::= ⟨*when*⟩ ⟨*event-expr*⟩ ⟨*action-expr*⟩
  | ⟨*when-expr*⟩ 'THEN' ⟨*action-expr*⟩

⟨*when*⟩ ::= 'WHEN' | 'WHENEVER'

⟨*while-expr*⟩ ::= ⟨*while*⟩ ⟨*state-expr*⟩ ⟨*controller-expr*⟩

⟨*while*⟩ ::= 'WHILE' | 'WHILEVER'

⟨*event-expr*⟩ ::= *empty* | *event*
  | ⟨*op1-input*⟩ ⟨*event-expr*⟩
  | ⟨*event-expr*⟩ ⟨*op2-input*⟩ ⟨*event-expr*⟩
  | ⟨*event-expr*⟩ 'is' *constant*
  | ⟨*event-expr*⟩ 'and' ⟨*state-expr*⟩

⟨*action-expr*⟩ ::= *empty* | *action*
  | ⟨*action-expr*⟩ ⟨*op2-output*⟩ ⟨*action-expr*⟩
  | ⟨*action-expr*⟩ 'and' ⟨*controller-expr*⟩

⟨*state-expr*⟩ ::= *constant* | *state*
  | ⟨*op1-input*⟩ ⟨*state-expr*⟩
  | ⟨*event-expr*⟩ ⟨*op2-input*⟩ ⟨*state-expr*⟩
  | ⟨*state-expr*⟩ 'is' *constant*

⟨*controller-expr*⟩ ::= *constant* | *controller*
  | ⟨*action-expr*⟩ ⟨*op2-output*⟩ ⟨*controller-expr*⟩
  | 'repeatedly' ⟨*action-expr*⟩

⟨*input-op1*⟩ := 'not' | ...

⟨*input-op2*⟩ := 'or' | 'and' | ...

⟨*output-op2*⟩ := 'or' | 'and' | ...

Figure 1: SOBORO Syntax

support implementing the behaviors in the aforementioned abstract representations, however, SOBORO focuses on supporting the two dominant programming paradigms for social robot programmers: trigger-action and imperative [6, 10].

3 SOBORO DOMAIN-SPECIFIC LANGUAGE

SOBORO is a domain-specific language that its programs can be compiled into functional reactive programs. Based on our previous research on understanding the challenges with programming social robot behaviors [6, 10], we designed SOBORO to (1) enable programmers to effortlessly express complex interactive behaviors using both trigger-action and imperative programming paradigms and (2) have straightforward and reasonable semantics that enables analyzing pragmatically, if needed.

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 ⟨*behavior*⟩ definition describes an interactive behavior as a list of ⟨*rule*⟩-s, defining mappings between *events* and *actions* via ⟨*when-expr*⟩ and *states* and *controllers* via ⟨*while-expr*⟩. The ⟨*event-expr*⟩ defines the event composition expression and the ⟨*action-expr*⟩ defines the action composition expression; they let programmers express custom trigger events and complex activation signals, respectively. Similarly, the ⟨*state-expr*⟩ defines the state composition expression and the ⟨*controller-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 ⟨*state-expr*⟩ involving `and` and ⟨*action-expr*⟩ involving `and`, respectively. The term–`'repeatedly'` ⟨*action − expr*⟩–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 ⟨*when-expr*⟩ has finished successfully. The ⟨*when-expr*⟩ involving `'THEN'` is a convenient syntax that can be expressed with multiple ⟨*when-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.js[2]. The `interp` function implements the semantics of SOBORO syntax. For example, the function interprets the parts of the syntax tree corresponding to ⟨*when-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 ⟨*event-expr*⟩ and ⟨*when-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 RxJS[3] and implemented the compiler in JavaScript for explanatory purposes only. Other

[2] https://pegjs.org/
[3] https://rxjs.dev/

```
WHEN HumanSpeech is "hello robot"
Say "hello there!"
```

(a) An example SOBORO program.

```
{
  "type": "behavior",
  "value": [{
    "type": "rule",
    "value": {
      "type": "when-expr",
      "value": [{
        "type": "event-expr",
        "value": ["is", {
          "type": "event",
          "value": "HumanSpeech"
        }, "hello robot"]
      }, {
        ...
      }, 1, null],
    }
  }]
}
```

(b) An example abstract syntax tree representing the example program.

```
var behavior = function (inputs) {
  var events = inputs[0];
  var states = inputs[0];
  var actions = {
    Say: empty(),
  };
  var controllers = {};

  actions["Say"] = merge(  // merge a new action (2nd arg)
    actions["Say"],
    events["HumanSpeech"]
      .pipe(
        filter(function (val) {
          return val === "hello robot";
        })
      )
      .pipe(
        mapTo(of("hello there!")),  // map an event to an action value
        take(1)  // respond "tree.value[2]" times
      )
  );

  var outputs = [actions, controllers];
  return outputs;
};
```

(c) An example compiled program of the example program.

Figure 2: An example interactive behavior in the representations involved in the SOBORO compiler.

```
// prog: a string SOBORO program
// inOutDesc: a dictionary describing robot inputs and outputs
var compiler = function (progIn, inOutDesc) {
  var tree = parse(progIn);
  var progOut = interp(tree, inOutDesc);
  var progOut = format(progOut);  // indent the code, etc.
  return progOut;
}

...

// tree: an abstract syntax tree returned from parse
// inOutDesc: a dictionary describing robot inputs and outputs
function interp(tree, inOutDesc) {
  if (tree.type === "behavior") {
    return `var behavior = function (inputs) {
  ...
  return outputs;
}`;
      // ...
    // ...
  } else if (tree.type === "when-expr") {
    var actionDesc = interp(tree.value[0], inOutDesc);
    var event = interp(tree.value[0], inOutDesc);
    if (tree.value[3] === null) {
      if (actionDesc.length === 1) {
        return `actions["${actionDesc[0].name}"] = merge(  // merge a new action (2nd arg)
  actions["${actionDesc[0].name}],
  ${event}.pipe(
    mapTo(of(${actionDesc[0].value})),  // map an event to an action value
    take(${tree.value[2]})  // respond "tree.value[2]" times
  )
);`;
        }
      }
      // ...
  } else if (tree.type === "event-expr") {
    if (tree.op === "is") {
      return `${tree.value[0]}.pipe(
  filter(function(val) {
    return val === ${tree.value[1]};
  })
)`;
        // ...
      // ...
    // ...
}
```

Figure 3: A snippet of an example compiler implementation.

data formats like YAML (Yet Another Markup Language), programming 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 idealized 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.
- *HumanSpeech* is an output of the speech recognizer.
- *Time* represents the time as an 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 following actions

- *Say* causes the robot to say a phrase.
- *PlaySound* starts playing the specified sound file.

and 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

4.1 Interactive Storytelling

```
w
```

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

```
// Scheduled meditation guide
WHENEVER Time is "8:00am"
PlaySound "morning_meditation_sound.mp3" or PlaySound
    "morning_meditation_instructions.mp3"

WHENEVER Time is "4:00pm"
PlaySound "afternoon_meditation_sound.mp3" or PlaySound
    "morning_meditation_instructions.mp3"

// On-demand meditation guide
WHENEVER HumanSpeech is "play background music"
repeatedly PlaySound "background_meditation_sound.mp3"

WHENEVER HumanSpeech is "stop"
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[5]. As SOBORO becomes more complex, it may difficult to extend the current natural language-like format. We plan to investigate adopting a data format like JSON, the format used by Vega-lite [15]–a visualization tool targeting a similar user group (e.g., programmers). Currently, SOBORO allows programmers to create multiple rules that can be triggered by the same input event (or state), which is not the desired behavior. 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.

[5] https://github.com/superscriptjs/superscript/wiki

REFERENCES

[1] Anki. 2022. Cozmo SDK. http://cozmosdk.anki.com/docs/. Accessed: 2022-02-14.

[2] Emilia I Barakova, Jan CC Gillesen, Bibi EBM Huskens, and Tino Lourens. 2013. End-user programming architecture facilitates the uptake of robots in social therapies. *Robotics and Autonomous Systems* 61, 7 (2013), 704–713.

[3] Jonathan Bohren and Steve Cousins. 2010. The SMACH high-level executive [ROS news]. *Robotics & Automation Magazine* 17, 4 (2010), 18–20.

[4] Crystal Chao and Andrea L Thomaz. 2012. Timing in multimodal turn-taking interactions: Control and analysis using timed petri nets. *Journal of Human-Robot Interaction* 1, 1 (2012), 4–25.

[5] Michael Jae-Yoon Chung, Justin Huang, Leila Takayama, Tessa Lau, and Maya Cakmak. 2016. Iterative design of a system for programming socially interactive service robots. In *International Conference on Social Robotics*. 919–929.

[6] Michael Jae-Yoon Chung, Mino Nakura, Sai Harshita Neti, Anthony Lu, Elana Hummel, and Maya Cakmak. 2020. ConCodeIt! A Comparison of Concurrency Interfaces in Block-Based Visual Robot Programming. In *International Conference on Robot and Human Interactive Communication*. IEEE, 245–252.

[7] Michele Colledanchise and Petter Ögren. 2016. How behavior trees modularize hybrid control systems and generalize sequential behavior compositions, the subsumption architecture, and decision trees. *Transactions on robotics* 33, 2 (2016), 372–389.

[8] Dylan F Glas, Takayuki Kanda, and Hiroshi Ishiguro. 2016. Human-robot interaction design using Interaction Composer eight years of lessons learned. In *International Conference on Human-Robot Interaction*. ACM/IEEE, 303–310.

[9] Matthew Huggins, Anastasia K Ostrowski, Andrew Rapo, Eric Woudenberg, Cynthia Breazeal, and Hae Won Park. 2021. The Interaction Flow Editor: A New Human-Robot Interaction Rapid Prototyping Interface. *arXiv preprint arXiv:2108.13838* (2021).

[10] Rajeswari Hita Kambhamettu, Michael Jae-Yoon Chung, Vinitha Ranganeni, and Patrıcia Alves-Oliveira. 2021. Collecting Insights into How Novice Programmers Naturally Express Programs for Robots. In *Workshop on the intersection of HCI and PL*.

[11] Tino Lourens and Emilia Barakova. 2011. User-friendly robot environment for creation of social scenarios. In *International Work-Conference on the Interplay between Natural and Artificial Computation*. 212–221.

[12] MYSTYROBOTICS. 2022. MYSTYROBOTICS SDK. http://sdk.mistyrobotics.com/. Accessed: 2022-02-14.

[13] Hugo Pacheco and Nuno Macedo. 2020. ROSY: An elegant language to teach the pure reactive nature of robot programming. In *International Conference on Robotic Computing*. IEEE, 240–247.

[14] Emmanuel Pot, Jérôme Monceaux, Rodolphe Gelin, and Bruno Maisonnier. 2009. Choregraphe: a graphical tool for humanoid robot programming. In *The International Symposium on Robot and Human Interactive Communication*. IEEE, 46–51.

[15] Arvind Satyanarayan, Dominik Moritz, Kanit Wongsuphasawat, and Jeffrey Heer. 2016. Vega-lite: A grammar of interactive graphics. *Transactions on visualization and computer graphics* 23, 1 (2016), 341–350.

[16] temi. 2022. temi SDK. https://github.com/robotemi/sdk. Accessed: 2022-02-14.