SOBORO: A Social Robot Behavior Authoring Language

Michael Jae-Yoon Chung and Maya Cakmak

HRI22 PD/EUP Workshop (Paper), 2022/03/07

The Problem

- Social robots need **contents**
- End-user programming systems are **not expressive enough**
  - E.g., flow chart and block-based visual programming interfaces
- Programming a social robot is difficult **even for programmers**
  - Interactive behaviors are **multi-modal**
  - **No API standard** for programs with concurrency

SOcial RoBOt BehavioR AuthOring (SOBORO)

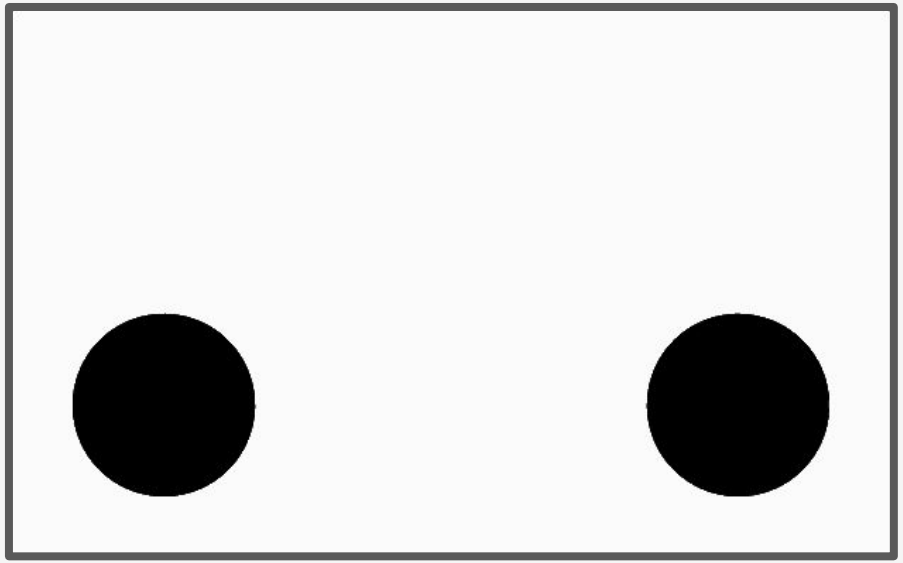
We propose a domain-specific language targeting **programmers** to author interactive behaviors authoring via **declarative specifications**.

The two key features:

1. **Imperative and reactive** programming friendly syntax
2. **Language-agnostic** compiler, outputting functional reactive programs[1]

[1]For a gentle introduction to reactive programming, see this tutorial.

The Robot



https://github.com/mjyc/tablet-robot-face

Inputs

Events

- Ready
- HumanSpeech
- Time

States

- HumanFace

Outputs

Actions

- Say
- PlaySound

Controllers

- SetImageTo
- SetEyePosX
- SetEyePosY

Example 1: Interactive Storytelling

```
// Storytelling behavior
WHEN ReadyChange
Say "Brown bear, brown bear, what do you see?" and SetImageTo "br
THEN SetImageTo "redbird.png"

WHEN HumanSpeech is "red bird" and HumanFace is "visible"
Say "red bird, red bird, what do you see?"
THEN SetImageTo "yellowduck.png"

WHEN HumanSpeech is "yellow duck" and HumanFace is "visible"
Say "yellow duck, yellow duck, what do you see?"
THEN SetImageTo "bluehorse.png"

// Gaze behavior
WHILEVER HumanFace is "visible"
SetEyePosX HumanFacePosX and SetEyePosX HumanFacePosY

WHILEEVER HumanFace is "invisible"
SetEyePosX 0 and SetEyePosX 0
```

Inputs

Events

- Ready
- HumanSpeech
- Time

States

- HumanFace

Outputs

Actions

- Say
- PlaySound

Controllers

- SetImageTo
- SetEyePosX
- SetEyePosY

Example 1: Interactive Storytelling

```
// Storytelling behavior
WHEN ReadyChange
Say "Brown bear, brown bear, what do you see?" and SetImageTo "br
THEN SetImageTo "redbird.png"

WHEN HumanSpeech is "red bird" and HumanFace is "visible"
Say "red bird, red bird, what do you see?"
THEN SetImageTo "yellowduck.png"

WHEN HumanSpeech is "yellow duck" and HumanFace is "visible"
Say "yellow duck, yellow duck, what do you see?"
THEN SetImageTo "bluehorse.png"

// Gaze behavior
WHILEVER HumanFace is "visible"
SetEyePosX HumanFacePosX and SetEyePosX HumanFacePosY

WHILEEVER HumanFace is "invisible"
SetEyePosX 0 and SetEyePosX 0
```

Syntax

```
⟨behavior⟩ ::= '[' ⟨rule⟩, ⟨rule⟩, ... ']'
⟨rule⟩ ::= ⟨when-expr⟩ | ⟨while-expr⟩
⟨when-expr⟩ ::= ⟨when⟩ ⟨event-expr⟩ ⟨action-expr⟩
    | ⟨when-expr⟩ 'THEN' ⟨action-expr⟩
⟨while-expr⟩ ::= ⟨while⟩ ⟨state-expr⟩ ⟨controller-expr⟩
⟨event-expr⟩ ::= empty | event
    | ⟨op1-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⟩
...
```

Example 1: Interactive Storytelling

```
// Storytelling behavior
WHEN ReadyChange
Say "Brown bear, brown bear, what do you see?" and SetImageTo "br
THEN SetImageTo "redbird.png"

WHEN HumanSpeech is "red bird" and HumanFace is "visible"
Say "red bird, red bird, what do you see?"
THEN SetImageTo "yellowduck.png"

WHEN HumanSpeech is "yellow duck" and HumanFace is "visible"
Say "yellow duck, yellow duck, what do you see?"
THEN SetImageTo "bluehorse.png"

// Gaze behavior
WHILEVER HumanFace is "visible"
SetEyePosX HumanFacePosX and SetEyePosX HumanFacePosY

WHILEEVER HumanFace is "invisible"
SetEyePosX 0 and SetEyePosX 0
```

TAP-like syntax for reactive programming

Syntax

```
⟨behavior⟩ ::= '[' ⟨rule⟩, ⟨rule⟩, ... ']'
⟨rule⟩ ::= ⟨when-expr⟩ | ⟨while-expr⟩
⟨when-expr⟩ ::= ⟨when⟩ ⟨event-expr⟩ ⟨action-expr⟩
    | ⟨when-expr⟩ 'THEN' ⟨action-expr⟩
⟨while-expr⟩ ::= ⟨while⟩ ⟨state-expr⟩ ⟨controller-expr⟩
⟨event-expr⟩ ::= empty | event
    | ⟨op1-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⟩
...
```

Example 1: Interactive Storytelling

```
// Storytelling behavior
WHEN ReadyChange
Say "Brown bear, brown bear, what do you see?" and SetImageTo "br
THEN SetImageTo "redbird.png"

WHEN HumanSpeech is "red bird" and HumanFace is "visible"
Say "red bird, red bird, what do you see?"
THEN SetImageTo "yellowduck.png"

WHEN HumanSpeech is "yellow duck" and HumanFace is "visible"
Say "yellow duck, yellow duck, what do you see?"
THEN SetImageTo "bluehorse.png"

// Gaze behavior
WHILEVER HumanFace is "visible"
SetEyePosX HumanFacePosX and SetEyePosX HumanFacePosY

WHILEEVER HumanFace is "invisible"
SetEyePosX 0 and SetEyePosX 0
```

Precise definition via explicit typing
Multi-modal & temporal interaction

Syntax

⟨behavior⟩ ::= '[' ⟨rule⟩, ⟨rule⟩, ... ']'
⟨rule⟩ ::= ⟨when-expr⟩ | ⟨while-expr⟩
⟨when-expr⟩ ::= ⟨when⟩ ⟨event-expr⟩ ⟨action-expr⟩
    | ⟨when-expr⟩ 'THEN' ⟨action-expr⟩
⟨while-expr⟩ ::= ⟨while⟩ ⟨state-expr⟩ ⟨controller-expr⟩
⟨event-expr⟩ ::= *empty* | *event*
    | ⟨op1-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⟩
...

Example 1: Interactive Storytelling

Triggers once

```
// Storytelling behavior
WHEN ReadyChange
Say "Brown bear, brown bear, what do you see?" and SetImageTo "br
THEN SetImageTo "redbird.png"

WHEN HumanSpeech is "red bird" and HumanFace is "visible"
Say "red bird, red bird, what do you see?"
THEN SetImageTo "yellowduck.png"

WHEN HumanSpeech is "yellow duck" and HumanFace is "visible"
Say "yellow duck, yellow duck, what do you see?"
THEN SetImageTo "bluehorse.png"

// Gaze behavior
WHILEVER HumanFace is "visible"
SetEyePosX HumanFacePosX and SetEyePosX HumanFacePosY

WHILEEVER HumanFace is "invisible"
SetEyePosX 0 and SetEyePosX 0
```

Always trigger

Convenient syntax and semantics for sequencing actions

Syntax

```
⟨behavior⟩ ::= '[' ⟨rule⟩, ⟨rule⟩, ... ']'
⟨rule⟩ ::= ⟨when-expr⟩ | ⟨while-expr⟩
⟨when-expr⟩ ::= ⟨when⟩ ⟨event-expr⟩ ⟨action-expr⟩
    | ⟨when-expr⟩ 'THEN' ⟨action-expr⟩
⟨while-expr⟩ ::= ⟨while⟩ ⟨state-expr⟩ ⟨controller-expr⟩
⟨event-expr⟩ ::= empty | event
    | ⟨op1-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⟩
...
```

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;
}

...
```

Example SOBORO Program

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

Abstract Syntax Tree

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

Compiled Reactive 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;
};
```

Example SOBORO Program

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

Abstract Syntax Tree

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

Could be in YAML

Compiled Reactive 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;
};
```

The data format, language, and reactive library choices are not required by SOBORO

Could be in Python

The `interp` function

```
// tree: an abstract syntax tree returned from parse
// inOutDesc: a dictionary describing robot inputs and outputs
function interp(tree, inOutDesc) {
  if (tree.type === "behavior") {
    // ...
  // ...
  } else if (tree.type === "when-expr") {
    var actionDesc = interp(tree.value[0], inOutDesc);  // create a new event
    var event = interp(tree.value[0], inOutDesc);  // create a new event
    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
  )
);`;
```

Future Work

1. **Variable and composition** by leveraging solutions used in chatbot script
    - E.g., superscript.js
2. A **different data format** than the natural language like text format
    - E.g., JSON which Vega-lite
3. **Developer tools** such as program verifier
    - E.g., to prevent undesirable behaviors at compile time
4. High-level **interaction grammar** design
    - E.g., based on findings from the past HRI, HCI research

Summary

- Presented SOBORO DSL
- Imperative and reactive programming friendly syntax
- Language agnostic compiler
- Future work

Thank you! Any questions?