REACTIVE PROGRAMMING FOR ROBOT APPLICATIONS

MICHAEL JAE-YOON CHUNG

ROS SEATTLE MEETUP, 2019/03/27


TUG
DASH
PAL ROBOTICS


12:45
Dinner with Judy
CHELSEA MARKET MAKERS


relay

CHALLENGES

- Uncertainty

CHALLENGES

- Uncertainty
- Temporailty

CHALLENGES

- Uncertainty
- Temporailty
- Concurrency

CHALLENGES

- Uncertainty
- Temporailty
- Concurrency

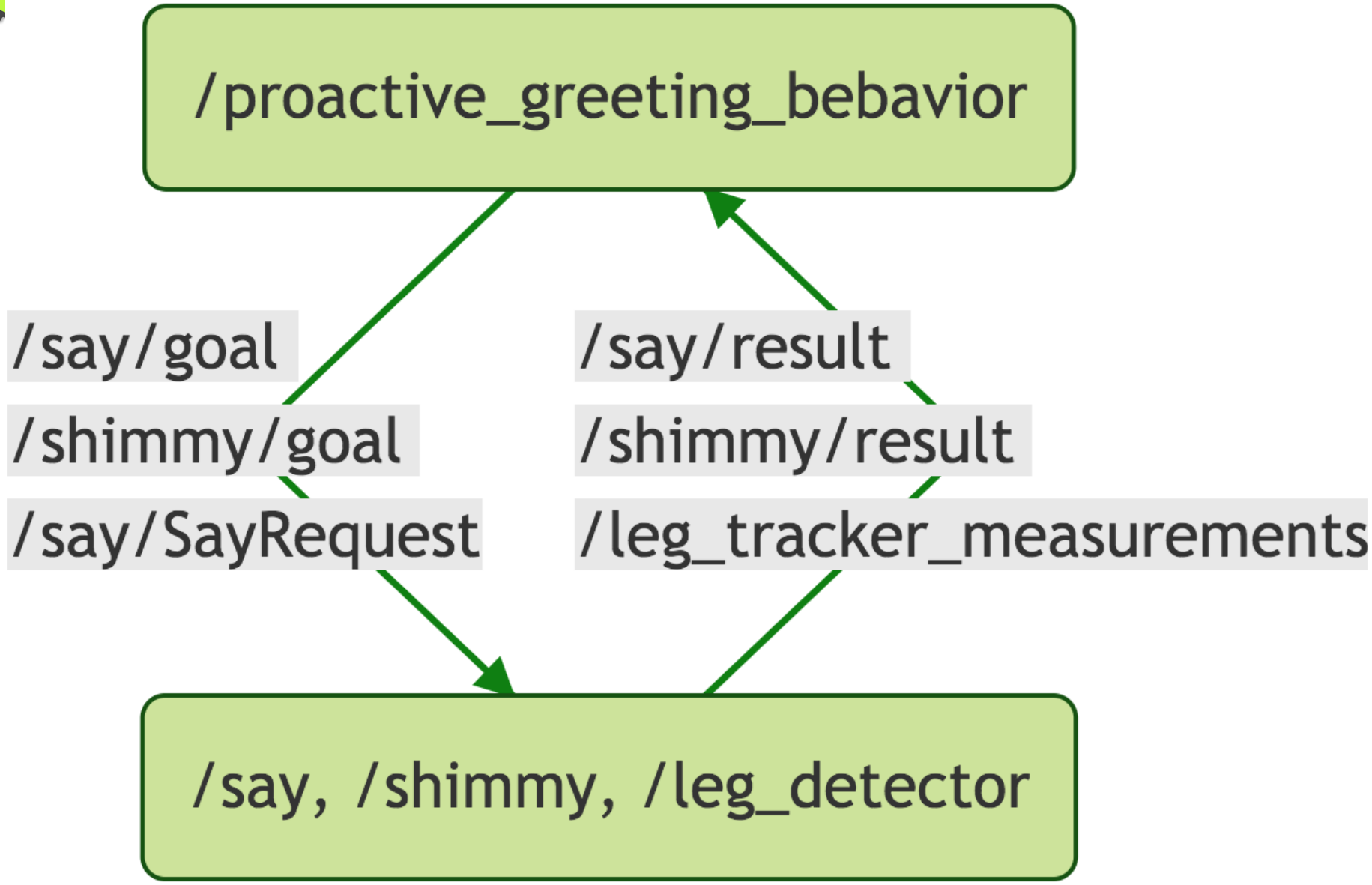
CHALLENGES

- Uncertainty
- Temporailty
- Concurrency

CYCLE.JS + ROS

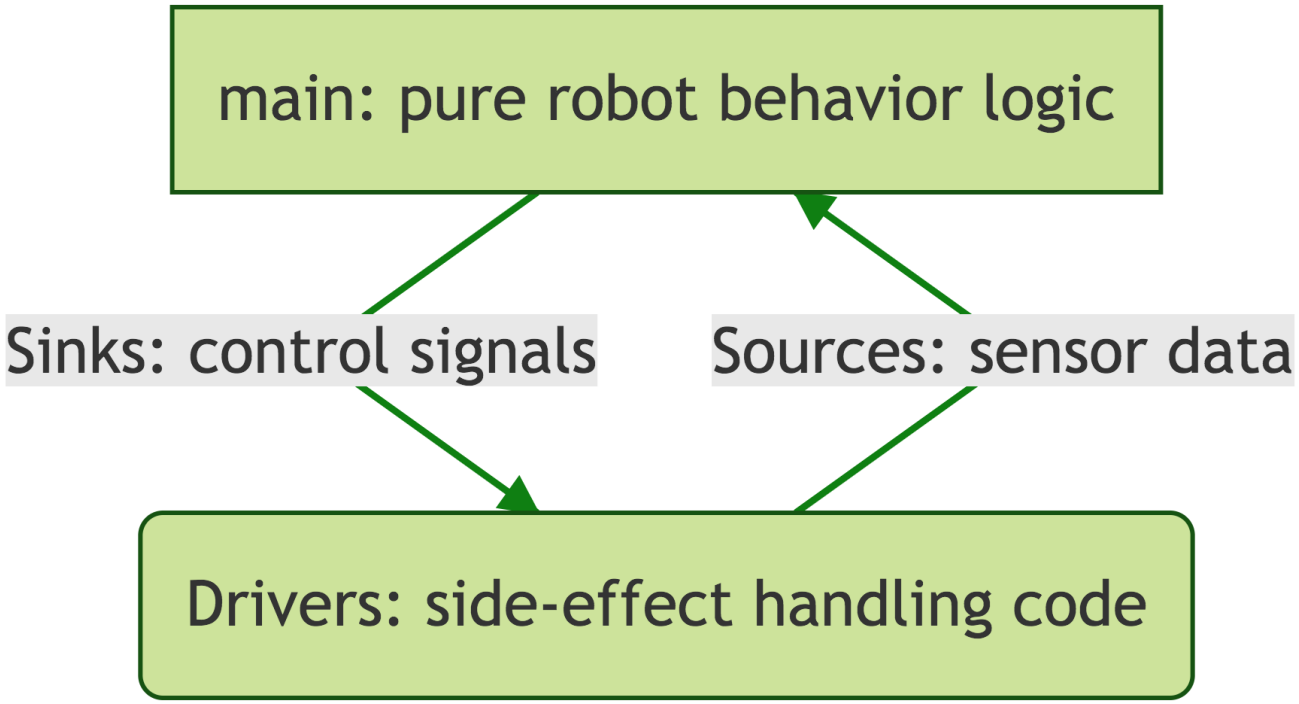
WHAT IS CYCLE.JS?

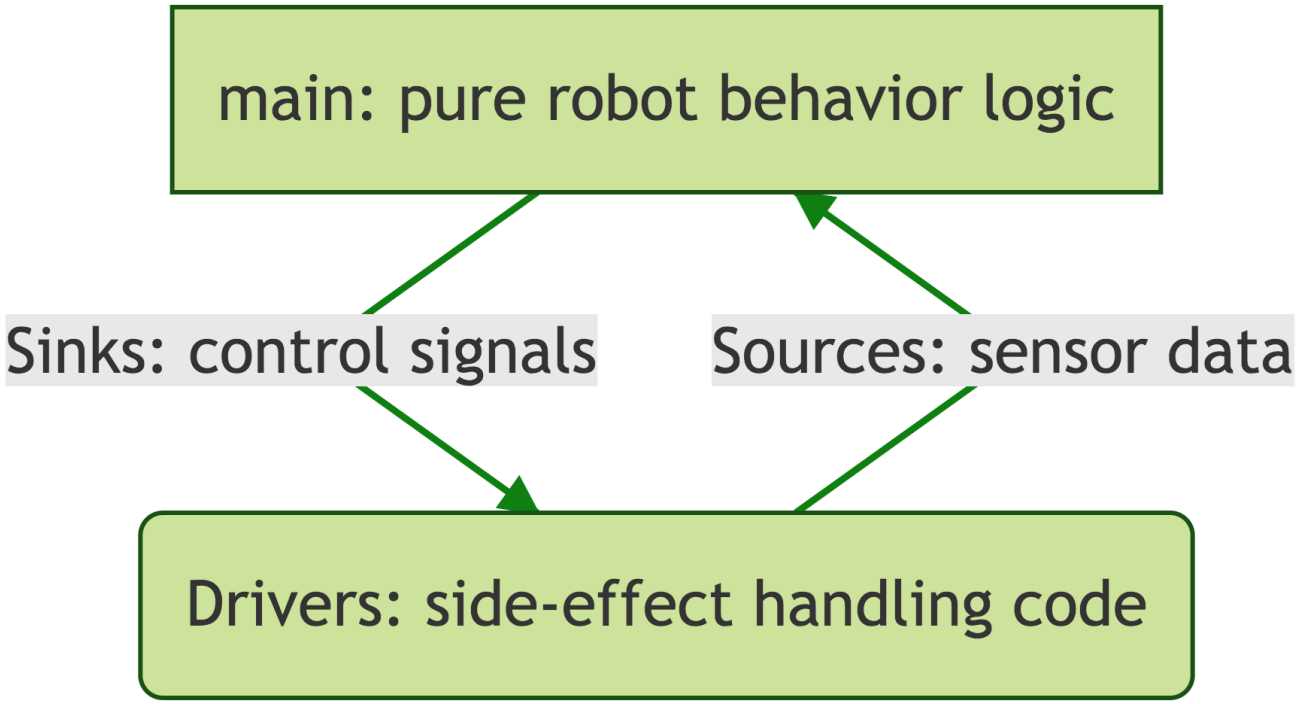
- functional and reactive programming framework in JavaScript
- abstraction that separates side-effect producing code from the main business logic code so the main code remains pure and predictable.


/proactive_greeting_bebavior
/say/goal
/shimmy/goal
/say/SayRequest
/say/result
/shimmy/result
/leg_tracker_measurements
/say, /shimmy, /leg_detector

New specifications:

- notify the server on certain events
- coordinate text-to-speech and base-movement


main: pure robot behavior logic
Sinks: control signals
Sources: sensor data
Drivers: side-effect handling code




```
sinks = main(sources)

sources = Drivers(sinks)
```

ROS APIS AS CYCLE.JS DRIVERS

```
{subTopics} = ROSTopicDriver({pubTopics});

{resps} = ROSServiceDriver({reqs});

{latestVals} = ROSParamDriver({newVals});
```

See cycle-ros-example github repo for details.

proactiveGreetingsApp.js

```
1   import xs from 'xstream';
2   ...
3
4   function main(sources) {
5     var actionGoals$ = sources.ROSTopic['/leg_tracker_measurements']
6       .map(function(data) { ... })  // transform data to distance
7       .filter(function(dist) { ... })  // discard out-of-range
8       .map(function(filteredDist) {
9         if (filteredDist < 0.5) {
10          return {
11            id: Date.now(),
12            say: 'How can I help you?',
13            shimmy: true,
14          }
15        } else {
16          return {
17            id: Date.now(),
18            say: 'Hi',
19            shimmy: false,
20          }
21        }
22      });
23
24    var notifyGoal$ = xs.combine(  // i.e., wait for two action results
25      sources.ROSTopic['/say/result'],
26      sources.ROSTopic['/shimmy/result'],
27    ).filter(function([sayResult, shimmyResult]) {
28      // true if action ids are matching, i.e., both actions are done
29    })
```

ros-seattle-meetup-20190328 Editor Preview Both Edit on StackBlitz

Check out cycle-robot-drivers github repo for more drivers and example applications!

index.js

```
1   import xs from 'xstream';
2   import {div, makeDOMDriver} from '@cycle/dom';
3   import {run} from '@cycle/run';
4   import {makePoseDetectionDriver} from 'cycle-posenet-driver';
5   import {makeTabletFaceDriver} from
    '@cycle-robot-drivers/screen';
6   import {makeSpeechSynthesisDriver} from
    '@cycle-robot-drivers/speech';
7
8   function main(sources) {
9     var vdom$ = xs.combine(
10      sources.TabletFace.DOM,
11      sources.PoseDetection.DOM
12    ).map(function (vdoms) { return div(vdoms); });
13
14    var face$ = sources.PoseDetection.poses
15      .filter(
16        function(poses) {
```

https://ros-seattle-meetup-2019...

Console

ros-seattle-meetup-20190328 Editor Preview Both Edit on StackBlitz

RELATED WORK

- Functional Reactive Animation - ICFP97
- Yampa - 2002
- "Reactive ROS" topic in ROS discourse
- Playful

CONCLUSION

- Cycle.js + ROS as a reactive programming solution for robot applications
- The functional reactive programming and the side-effect separation (i.e., ports and adapters pattern) can be applied without Cycle.js
- Hope to see the ROS users adapting the patterns in python or cpp via RxPY or RxCpp

MORE READING

- Programming a social robot using Cycle.js
- Implementing a finite state machine in Cycle.js