Advanced Join Patterns for the Actor Model based on CEP Techniques

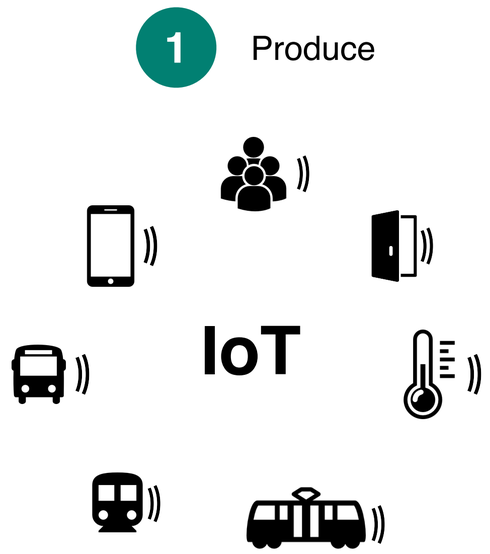
<Programming>

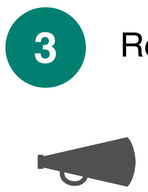
March 2021

Humberto Rodríguez A.
Joeri De Koster
Wolfgang De Meuter

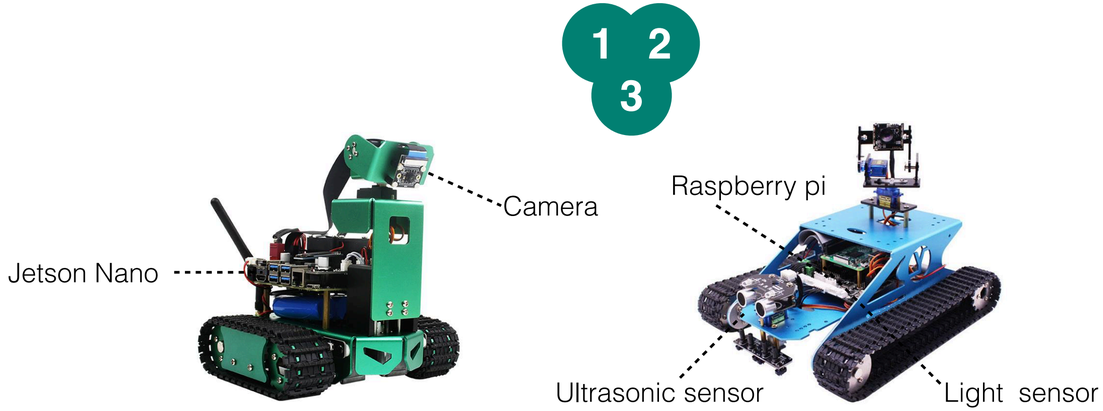
Reactive Applications

(A) Distributed





(B) Embedded



Robots based single-board computer (SBC)

Limited Interation Patterns

```elixir
def loop({ts_a, ts_b}) do
  state =
    receive do
      {:msg_a, timestamp} ->
       {timestamp, ts_b}

      {:msg_b, timestamp} ->
       {ts_a, timestamp}

      {:msg_c, timestamp} ->
       if ts_b > ts_a do
         # reaction code
       end
       {0,0} # reset state

    end # receive-end

   loop(state)

end
```

Example of how to detect a sequence of messages in Elixir

(MsgA → MsgB → MsgC)

Motivation: Smart-home scenario

[A1] Turn on the lights of a room if someone enters in it, and its ambient light is less than 40 lux.

[A2] Turn off the lights of a room after two minutes without detecting any movement.

[A3] Send me a notification when a window has been open for an hour.

[A4] Send a notification if someone presses the doorbell, but do not send a new notification after every doorbell press. Each notification must have an interval of at least 30 seconds.

[A5] Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave.

[A6] Fire a notification if the electricity consumption at home is greater than 200 kWh in the last three weeks.

[A7] Send a notification if the boiler fires three Floor Heating Failures and one Internal Failure within one hour. Each notification must have an interval of at least 60 minutes.

Online Poll

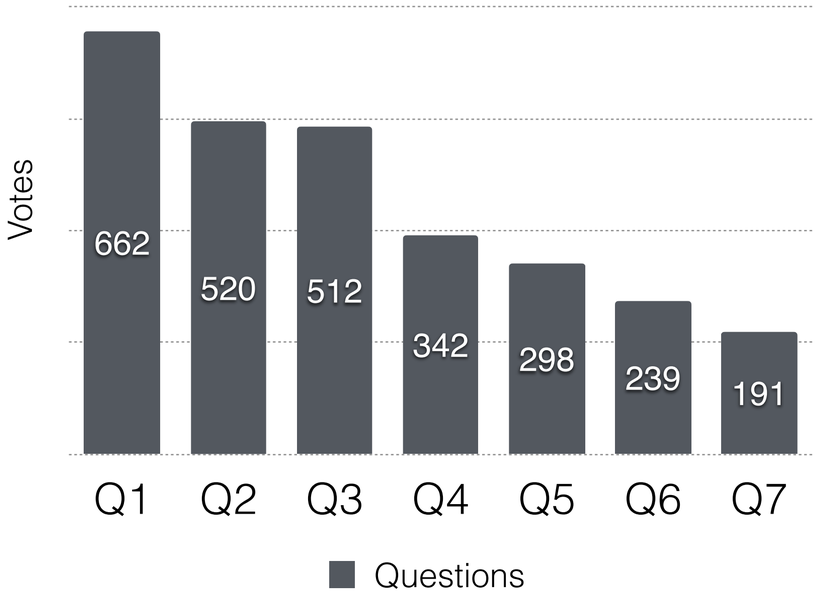
Automations ≈ Questions

I have automations that involve multiple devices and conditions. For example, `Turn on the lights of a room IF motion is detected AND its ambient light is LESS THAN 40 lux.` 90%

714 voters

29 countries

30 days



openHAB empowering the smart home https://doi.org/10.5281/zenodo.3666325

Home Assistant http://doi.org/10.5281/zenodo.3465385

Hubitat ELEVATE YOUR ENVIRONMENT http://doi.org/10.5281/zenodo.3464966

SmartThings http://doi.org/10.5281/zenodo.3464952

Correlation Requirements

1.Advanced filter mechanism
- Content-based
- Time-based

2.Flexible event selection policy
- First-in
- Last-in
- Nth-in
- For-all

3.Extensive correlation operators
- Conjunctions
- Disjunctions
- Sequencing
- Negation

4.Event accumulation
- Count-based
- Time-based

5.Event transformation
- Aggregation

[A1] Turn on the lights of a room if someone enters in it, and its ambient light is less than 40 lux.

[A2] Turn off the lights of a room after two minutes without detecting any movement.

[A3] Send me a notification when a window has been open for an hour.

[A4] Send a notification if someone presses the doorbell, but do not send a new notification after every doorbell press. Each notification must have an interval of at least 30 seconds.

[A5] Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave.

[A6] Fire a notification if the electricity consumption at home is greater than 200 kWh in the last three weeks.

[A7] Send a notification if the boiler fires three Floor Heating Failures and one Internal Failure within one hour. Each notification must have an interval of at least 60 minutes.

Sparrow

Domain-Specific Language for Coordinating Large Groups of Heterogeneous Actors

Language Abstractions as Macros

1. `pattern NAME as DEFINITION`
2. `reaction NAME do BODY end`
3. `react_to PATTERN_NAME , with: REACTION_NAME`
4. `remove_reaction REACTION_NAME, from: PATTERN_NAME`
5. `remove_all_reactions PATTERN_NAME`

Sparrow in a Nutshell

```
1  defmodule SmartHomeDemo do
2    use Sparrow.Actor
3
4    pattern motion as {:motion, id, :on, location}
5    pattern front_door_motion as motion{location= :front_door}
6    pattern entrance_hall_motion as motion{location= :entrance_hall, id~> mid}
7    pattern front_door_contact as {:contact, cid, :open, :front_door}
8
9    pattern occupied_home as front_door_motion and front_door_contact and entrance_hall_motion,
10                           options: [ interval: {60, :secs}, seq: true, last: true ]
11
12   pattern empty_home as entrance_hall_motion and front_door_contact and front_door_motion,
13                         options: [ interval: {60, :secs}, seq: true, last: true ]
14
15   reaction activate_home_scene(l, i, t), do: # code logic for arriving home
16   reaction activate_leave_scene(l, i, t), do:  # code logic for leaving home
17
18   react_to occupied_home, with: activate_home_scene
19   react_to empty_home, with: activate_leave_scene
20
21 end
```

Import language abstractions | Define patterns | Define reactions | Bind reactions to patterns

"Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave".

1.Advanced filter mechanism
- Content-based
- Time-based

2.Flexible event selection policy
- First-in
- Last-in
- Nth-in
- For-all

3.Extensive correlation operators
- Conjunctions
- Disjunctions
- Sequencing
- Negation

4.Event accumulation
- Count-based
- Time-based

5.Event transformation
- Aggregation

Sparrow in a Nutshell

```
1  defmodule SmartHomeDemo do
2    use Sparrow.Actor
3
4    pattern motion as {:motion, id, :on, location}
5    pattern front_door_motion as motion{location= :front_door}
6    pattern entrance_hall_motion as motion{location= :entrance_hall, id~> mid}
7    pattern front_door_contact as {:contact, cid, :open, :front_door}
8
9    pattern occupied_home as front_door_motion and front_door_contact and entrance_hall_motion,
10                             options: [ interval: {60, :secs}, seq: true, last: true ]
11
12   pattern empty_home as entrance_hall_motion and front_door_contact and front_door_motion,
13                           options: [ interval: {60, :secs}, seq: true, last: true ]
14
15   reaction activate_home_scene(l, i, t), do: # code logic for arriving home
16   reaction activate_leave_scene(l, i, t), do:  # code logic for leaving home
17
18   react_to occupied_home, with: activate_home_scene
19   react_to empty_home, with: activate_leave_scene
20
21 end
```

Import language abstractions | Define patterns | Define reactions | Bind reactions to patterns

"Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave".

occupied-home = FrontDoorMotion**ON** -> FrontDoorContact**Open** -> EntranceHallMotion**ON**

60 secs

Sparrow in a Nutshell

```
1  defmodule SmartHomeDemo do
2    use Sparrow.Actor
3
4    pattern motion as {:motion, id, :on, location}
5    pattern front_door_motion as motion{location= :front_door}
6    pattern entrance_hall_motion as motion{location= :entrance_hall, id~> mid}
7    pattern front_door_contact as {:contact, cid, :open, :front_door}
8
9    pattern occupied_home as front_door_motion and front_door_contact and entrance_hall_motion,
10                            options: [ interval: {60, :secs}, seq: true, last: true ]
11
12   pattern empty_home as entrance_hall_motion and front_door_contact and front_door_motion,
13                          options: [ interval: {60, :secs}, seq: true, last: true ]
14
15   reaction activate_home_scene(l, i, t), do: # code logic for arriving home
16   reaction activate_leave_scene(l, i, t), do:  # code logic for leaving home
17
18   react_to occupied_home, with: activate_home_scene
19   react_to empty_home, with: activate_leave_scene
20
21 end
```

Import language abstractions | Define patterns | Define reactions | Bind reactions to patterns

"Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave".

occupied-home = FrontDoorMotion**ON** -> FrontDoorContact**Open** -> EntranceHallMotion**ON**

60 secs

Elementary pattern

$$\mathrm{P}\langle N, S, O^{?}, G^{?}, R* \rangle$$

- **N**ame
- **S**elector $\mathrm{S}\langle type, attr1,..,attrN \rangle$
- **O**perators $\mathrm{O}\langle o^{+} \rangle$
- **G**uards $\mathrm{G}\langle g^{+} \rangle$

Sparrow in a Nutshell

```
1  defmodule SmartHomeDemo do
2    use Sparrow.Actor
3
4    pattern motion as {:motion, id, :on, location}
5    pattern front_door_motion as motion{location= :front_door}
6    pattern entrance_hall_motion as motion{location= :entrance_hall, id~> mid}
7    pattern front_door_contact as {:contact, cid, :open, :front_door}
8
9    pattern occupied_home as front_door_motion and front_door_contact and entrance_hall_motion,
10                            options: [ interval: {60, :secs}, seq: true, last: true ]
11
12   pattern empty_home as entrance_hall_motion and front_door_contact and front_door_motion,
13                          options: [ interval: {60, :secs}, seq: true, last: true ]
14
15   reaction activate_home_scene(l, i, t), do: # code logic for arriving home
16   reaction activate_leave_scene(l, i, t), do:  # code logic for leaving home
17
18   react_to occupied_home, with: activate_home_scene
19   react_to empty_home, with: activate_leave_scene
20
21 end
```

Import language abstractions | Define patterns | Define reactions | Bind reactions to patterns

"Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave".

occupied-home = FrontDoorMotion**ON** -> FrontDoorContact**Open** -> EntranceHallMotion**ON**

60 secs

Composite patterns

$\mathrm{P}\langle N, P_r, O^?, G^?, R* \rangle$ (first-order)

- **N**ame
- **P**attern **r**eference $\mathrm{P_r}\langle N \rangle$
- **O**perators $\mathrm{O}\langle o^+ \rangle$
- **G**uards $\mathrm{G}\langle g^+ \rangle$

Sparrow in a Nutshell

```
1  defmodule SmartHomeDemo do
2    use Sparrow.Actor
3
4    pattern motion as {:motion, id, :on, location}
5    pattern front_door_motion as motion{location= :front_door}
6    pattern entrance_hall_motion as motion{location= :entrance_hall, id~> mid}
7    pattern front_door_contact as {:contact, cid, :open, :front_door}
8
9    pattern occupied_home as front_door_motion and front_door_contact and entrance_hall_motion,
10                            options: [ interval: {60, :secs}, seq: true, last: true ]
11
12   pattern empty_home as entrance_hall_motion and front_door_contact and front_door_motion,
13                         options: [ interval: {60, :secs}, seq: true, last: true ]
14
15   reaction activate_home_scene(l, i, t), do: # code logic for arriving home
16   reaction activate_leave_scene(l, i, t), do:  # code logic for leaving home
17
18   react_to occupied_home, with: activate_home_scene
19   react_to empty_home, with: activate_leave_scene
20
21 end
```

Import language abstractions | Define patterns | Define reactions | Bind reactions to patterns

"Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave".

occupied-home = FrontDoorMotion**ON** -> FrontDoorContact**Open** -> EntranceHallMotion**ON**

60 secs

Composite patterns

$$\mathrm{P}\langle N, P_r, O^?, G^?, R* \rangle$$

- **N**ame
- **P**attern **r**eference $\mathrm{P_r}\langle N \rangle$
- **O**perators $\mathrm{O}\langle o^+ \rangle$
- **G**uards $\mathrm{G}\langle g^+ \rangle$

$$\mathrm{P}\langle N, F, F^+, O, G^?, R* \rangle$$

- Re**F**erence $\mathrm{F}\langle P_r, | P_a \rangle$

$\mathrm{P_a}\langle S, O^?, G^? \rangle$ Anonymous pattern

Sparrow in a Nutshell

```
1  defmodule SmartHomeDemo do
2    use Sparrow.Actor
3
4    pattern motion as {:motion, id, :on, location}
5    pattern front_door_motion as motion{location= :front_door}
6    pattern entrance_hall_motion as motion{location= :entrance_hall, id~> mid}
7    pattern front_door_contact as {:contact, cid, :open, :front_door}
8
9    pattern occupied_home as front_door_motion and front_door_contact and entrance_hall_motion,
10                             options: [ interval: {60, :secs}, seq: true, last: true ]
11
12   pattern empty_home as entrance_hall_motion and front_door_contact and front_door_motion,
13                           options: [ interval: {60, :secs}, seq: true, last: true ]
14
15   reaction activate_home_scene(l, i, t), do: # code logic for arriving home
16   reaction activate_leave_scene(l, i, t), do:  # code logic for leaving home
17
18   react_to occupied_home, with: activate_home_scene
19   react_to empty_home, with: activate_leave_scene
20
21 end
```

Import language abstractions | Define patterns | Define reactions | Bind reactions to patterns

"Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave".

occupied-home = FrontDoorMotion**ON** -> FrontDoorContact**Open** -> EntranceHallMotion**ON**

60 secs

Reactions

$$\mathrm{R}\langle N^{?}, L, I^{?}, T\rangle$$

- **N**ame
- **L**ist of messages
- Dictionary of **I**ntermediate transformation results
- Actor s**T**ate

Features supported by Sparrow patterns.

	Patterns		
	Elementary	Composite	Accumulation
Filtering			
Content-based	x	x	x
Time-based	x	x	x
Selection			
First-in	x	x	x
Last-in	x	x	x
Nth-in	x	x	x
For-all	x	x	x
Correlation			
Conjunction	-	x	x
Disjunction	-	x	x
Sequencing	-	x	x
Negation	x	x	x
Accumulation			
Count-based	-	-	x
Time-based	-	-	x
Transformation			
Aggregation	-	-	x

Evaluation

Smart-home Platforms

(Thread-based)

openHAB
empowering the smart home

- Rules DSL
- Jython

Home Assistant

- Python (AppDaemon)

Forum posts

Replies	Views	Likes
108	5508	82

Actor-based Language

elixir

Sparrow

- openHAB - https://doi.org/10.5281/zenodo.3611168
- Home-Assistant - http://doi.org/10.5281/zenodo.3611271

Automation #5 Implementation

“Activate the occupied-home scene when I arrive, and activate the empty-home scene when I leave”.

occupied-home = FrontDoorMotion**ON** -> FrontDoorContact**Open** -> EntranceHallMotion**ON**  60 secs

1 Jython
openHAB

```
1  from core.rules import rule
2  from core.triggers import when
3  from java.time import ZonedDateTime as ZDateTime
4
5  lastDoorOpen = ZDateTime.now().minusHours(24)
6  lastEHallMotion = ZDateTime.now().minusHours(24)
7  lastFDoorMotion = ZDateTime.now().minusHours(24)
8
9  @rule("(Py) Front Door Opened")
10 @when("Item Front_Door_Contact changed to OPEN")
11 def front_door_opened(event):
12     global lastDoorOpen
13     lastDoorOpen = ZDateTime.now()
14
15 @rule("(Py) Motion Detected - Entrance Hall")
16 @when("Item Entrance_Hall_Motion changed to ON")
17 def entrance_hall_motion(event):
18     global lastEHallMotion, lastFDoorMotion
19     lastEHallMotion = ZDateTime.now()
20
21     if lastFDoorMotion.isBefore(lastEHallMotion.minusSeconds(60)):
22         return
23
24     if lastEHallMotion.isAfter(lastDoorOpen) and lastDoorOpen.isAfter(lastFDoorMotion):
25         # code logic for arriving home
26
27 @rule("(Py) Motion Detected - Front Door")
28 @when("Item Front_Door_Motion changed to ON")
29 def front_door_motion(event):
30     global lastEHallMotion, lastFDoorMotion
31     lastFDoorMotion = ZDateTime.now()
32
33     if lastEHallMotion.isBefore(lastFDoorMotion.minusSeconds(60)):
34         return
35
36     if lastFDoorMotion.isAfter(lastDoorOpen) and lastDoorOpen.isAfter(lastEHallMotion):
37         # code logic for leaving home
```

2 Elixir

```
1  defmodule SmartHomeDemo do
2    require Timex
3
4    def loop({m_door, m_hall, c_door}) do
5      state =
6        receive do
7          {:motion, _id, :on, :front_door, m_door_dt} ->
8            if Timex.before?(Timex.shift(m_door_dt, seconds: -60), m_hall)  do
9              if Timex.after?(m_door_dt, c_door) and Timex.after?(c_door, m_hall) do
10               # code logic for leaving home
11             end
12           end
13           {m_door_dt, m_hall, c_door}
14
15         {:motion, _id, :on, :entrance_hall, m_hall_dt} ->
16           if Timex.before?(Timex.shift(m_hall_dt, seconds: -60), m_door)  do
17             if Timex.after?(m_hall_dt, c_door) and Timex.after?(c_door, m_door) do
18               # code logic for arriving home
19             end
20           end
21           {m_door, m_hall_dt, c_door}
22
23         {:contact, _id, :open, :front_door, dt} ->
24           {m_door, m_hall, dt}
25       end
26
27     loop(state)
28   end
29
30 end
```

3 Sparrow

```
1  defmodule SmartHomeDemo do
2    use Sparrow.Actor
3
4    pattern motion as {:motion, id, :on, location}
5    pattern front_door_motion as motion{location= :front_door}
6    pattern entrance_hall_motion as motion{location= :entrance_hall, id~> mid}
7    pattern front_door_contact as {:contact, cid, :open, :front_door}
8
9    pattern occupied_home as front_door_motion and front_door_contact and entrance_hall_motion,
10                           options: [ interval: {60, :secs}, seq: true, last: true ]
11
12   pattern empty_home as entrance_hall_motion and front_door_contact and front_door_motion,
13                         options: [ interval: {60, :secs}, seq: true, last: true ]
14
15   reaction activate_home_scene(l, i, t), do: # code logic for arriving home
16   reaction activate_leave_scene(l, i, t), do:  # code logic for leaving home
17
18   react_to occupied_home, with: activate_home_scene
19   react_to empty_home, with: activate_leave_scene
20
21 end
```

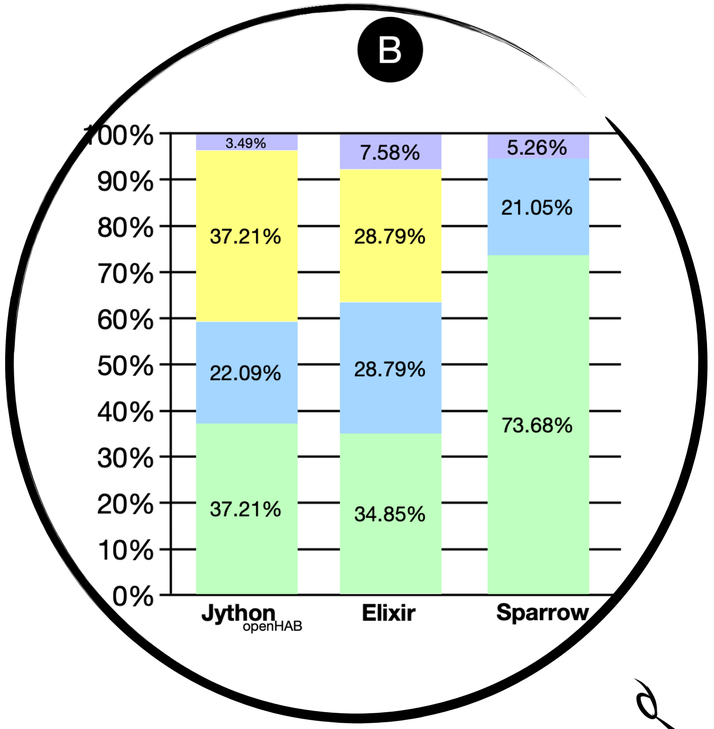
- Sequencing control
- State management
- Windowing management
- Correlation logic

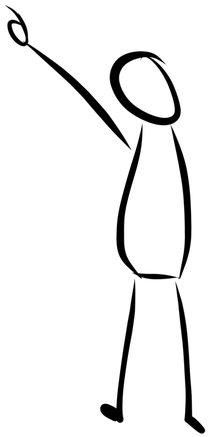
Implementation Statistics

A

	Jython openHAB	Elixir	Sparrow
Sequencing control	3	5	1
State management	32	19	0
Windowing management	19	19	4
Correlation logic	32	23	14
Total lines of code	86	66	19

B





Note: The results shown are the total LoC of the seven automation examples

Advanced Join Patterns for the Actor Model based on CEP Techniques

<Programming>

March 2021

Humberto Rodríguez Avila
Joeri De Koster
Wolfgang De Meuter