Orchestration of Web Applications with Hiphop.js

Colin Vidal – Manuel Serrano – Gérard Berry

firstName.lastName@inria.fr

Inria

November 28, 2017
The Web is a **standard** and **omnipresent** platform.

- Characterized by numerous actors (users, servers, online services, IoT devices, etc.) interacting by the mean of **asynchronous events**.

- The **orchestration** of Web applications is the harmonious interactions between those actors.
Context & Interest
Context & Interest

Events (user)

Led Zeppelin - Stairway To Heaven
7,883,000 views

user666
Published on Nov 8, 2016
Context & Interest

Events (user)

Events (multimedia)
Context & Interest

Events (user)

Events (multimedia)

Events (network)

All those events depend on states!
All those events depends of states!
Problem & Motivation

- JavaScript: the **mainstream language** to implement the Web (clients, servers, IoT devices).
- Event orchestration via an **asynchronous event-loop**.
Problem & Motivation

- JavaScript: the **mainstream language** to implement the Web (clients, servers, IoT devices).

- Event orchestration via an **asynchronous event-loop**.
Problem & Motivation

- JavaScript: the **mainstream language** to implement the Web (clients, servers, IoT devices).
- Event orchestration via an **asynchronous event-loop**.
Problem & Motivation

▶ JavaScript: the **mainstream language** to implement the Web (clients, servers, IoT devices).
▶ Event orchestration via an **asynchronous event-loop**.
Problem & Motivation

- Control-flow is implicit and non-deterministic: it is driven by event order.
- A global state automata is assumed by the event listener model.

The *orchestration with the event-loop* hard to verify and to maintain.

[Mikkonen and Taivalsaari, 2007, Kambona et al., 2013, Alan Jeffrey, 2015, ...]
Our proposal: to adapt reactive programming to the Web

A JavaScript domain-specific language based on temporal constructions:

- **Explicit control-flow** using sequence, preemption and parallelism operators.
- The program state is **in the code**.
Our proposal: to adapt reactive programming to the Web

A JavaScript domain-specific language based on temporal constructions:

- **Explicit control-flow** using sequence, preemption and parallelism operators.
- The program state is **in the code**.

Hiphop.js
Basics of Web programming

<html>
<body>
<div id="counter">0</div>
</body>
</html>
<html>
  <body>
    <button onclick=~{incrAndUpdate()}>increment and update label</button>
    <div id="counter">0</div>
  </body>
</html>
<html>
  <script>~{
    var cnt = 0;
    function incrAndUpdate() {
      cnt++;
      document.get("counter").innerHTML = cnt;
    }
  }
</script>
<body>
  <button onclick=~{incrAndUpdate()}>increment and update label</button>
  <div id="counter">0</div>
</body>
</html>
<html>
  <script>
    var cnt = 0;
    function incrAndUpdate() {
      cnt++;
      document.get("counter").innerHTML = cnt;
    }
  </script>
  <body>
    <button onclick=~{incrAndUpdate()}>increment and update label</button>
    <div id="counter">0</div>
  </body>
</html>
Basics of Hiphop.js: ABRO++ in the Web

<html>
<body>
<button onclick=~{}>a</button>
<button onclick=~{}>b</button>
<button onclick=~{}>r</button>
<div>o emitted <react~{} times</div>
</body>
</html>
Basics of Hiphop.js: ABRO++ in the Web

<html>

<body>

<button onclick=~{m.inputAndReact("a")}>a</button>
<button onclick=~{m.inputAndReact("b")}>b</button>
<button onclick=~{m.inputAndReact("r")}>r</button>

<div>o emitted <react>~{m.value.o}</react> times</div>

</body>

</html>
Basics of Hiphop.js: ABRO++ in the Web

```html
<html>
<script>~{
    var m = new ReactiveMachine(
        MODULE {
            IN a, b, r;
            OUT o(0);
            LOOPEACH (NOW(r)) {
                FORK {
                    AWAIT (NOW(a));
                }
                PAR {
                    AWAIT (NOW(b));
                }
                EMIT o(PREVAL(o) + 1);
            }
        }
    );
}
</script>
<body>
    <button onclick=~{m.inputAndReact("a")}>a</button>
    <button onclick=~{m.inputAndReact("b")}>b</button>
    <button onclick=~{m.inputAndReact("r")}>r</button>

    <div>o emitted <react>~{m.value.o}</react> times</div>
</body>
</html>
```
Basics of Hiphop.js: ABRO++ in the Web

```html
<html>
<script>~{
  var m = new ReactiveMachine(
    MODULE {
      IN a, b, r;
      OUT o(0);
      LOOPEACH (NOW (r)) {
        FORK {
          AWAIT (NOW (a));
        }
        PAR {
          AWAIT (NOW (b));
        }
        EMIT o (PREVAL (o) + 1);
      }
    }
  );
}</script>
<body>
  <button onclick=~{m.inputAndReact("a")}>a</button>
  <button onclick=~{m.inputAndReact("b")}>b</button>
  <button onclick=~{m.inputAndReact("r")}>r</button>
  <div>o emitted <react>~{m.value.o}</react> times</div>
</body>
</html>
```
Execution model

Before the JavaScript runtime

Translation of Hiphop.js statement into a JavaScript value.

```javascript
var m = new ReactiveMachine(
    MODULE { ... }
);
```

Preprocessor

```javascript
var m = new ReactiveMachine(
    value
);
```

foo.js

foo.out.js
Execution model

At the JavaScript runtime

**Execution** of a new ReactiveMachine *compiles* a Hiphop.js program and **returns a JavaScript object** containing:

- A simulation of the Boolean circuit implementing the program.
- A JavaScript API allowing programmer to interact with the program (**inputAndReact**, for instance).

**Implementation**

The whole toolchain (preprocessor included) is written in JavaScript and is a free software.
function makePause(n) {
    if (n <= 0) {
        return NOTHING;
    } else if (n == 1) {
        return PAUSE;
    } else {
        return SEQUENCE {
            PAUSE;
            ${makePause(n - 1)};
        }
    }

}
function makePause(n) {
    if (n <= 0) {
        return NOTHING;
    } else if (n == 1) {
        return PAUSE;
    } else {
        return SEQUENCE {
            PAUSE;
            ${makePause(n - 1)};
        }
    }
}

MODULE {
    OUT a, b;
    EMIT a;
    ${makePause(2)};
    EMIT b;
}
Embedding with JavaScript: building programs on-the-fly

```javascript
function makePause(n) {
    if (n <= 0) {
        return NOTHING;
    } else if (n == 1) {
        return PAUSE;
    } else {
        return SEQUENCE {
            PAUSE;
            ${makePause(n - 1)};
        }
    }
}

MODULE {
    OUT a, b;
    EMIT a;
    ${makePause(2)};
    EMIT b;
}
```
Embedding with JavaScript: building programs on-the-fly

```javascript
function makePause(n) {
    if (n <= 0) {
        return NOTHING;
    } else if (n == 1) {
        return PAUSE;
    } else {
        return SEQUENCE {
            PAUSE;
            ${makePause(n - 1)};
        }
    }
}

MODULE {
    OUT a, b;
    EMIT a;
    ${makePause(2)};
    EMIT b;
}
```

JavaScript runtime

```javascript
SEQUENCE {
    PAUSE;
    SEQUENCE {
        PAUSE;
        NOTHING;
    }
}
```

```javascript
MODULE {
    OUT a, b;
    EMIT a;
    SEQUENCE {
        PAUSE;
        SEQUENCE {
            PAUSE;
            NOTHING;
        }
    }
    EMIT b;
}
```
function makePause(n) {
  if (n <= 0) {
    return NOTHING;
  } else if (n == 1) {
    return PAUSE;
  } else {
    return SEQUENCE {
      PAUSE;
      ${makePause(n - 1)};
    };
  }
}

MODULE {
  OUT a, b;
  EMIT a;
  ${makePause(2)};
  EMIT b;
}
Hiphop.js expressions extends JavaScript expressions with signal accessors.

\[ \text{VAL(player).cnt > document.get("cnt") && NOW(stop)} \]
Hiphop.js expressions extends JavaScript expressions with signal accessors.

\[
\text{VAL}(\text{player}).\text{cnt} > \text{document.get("cnt")} \land \text{NOW}(\text{stop})
\]
Orchestrate actions taking (logical) time

VAR player = new Audio();
EXEC (player.src = VAL(src),
   player.play(),
   player.addEventListener("ended", function() {
      MACHINE.inputAndReact(EXECID)}))

ON SUSP player.pause()
ON RES player.play()
ON KILL player.stop();
EMIT over;

Control reach this point react after the calling
MACHINE.inputAndReact(EXECID)...

13 / 16
Orchestrate actions taking (logical) time

SUSPEND TOGGLE (NOW (playPause)) {
  VAR player = new Audio();
  EXEC (player.src = VAL(src),
        player.play(),
        player.addEventListener("ended", function() {
          MACHINE.inputAndReact(EXECID))})
  ONSUSP player.pause()
  ONRES player.play()
  ONKILL player.stop();
  EMIT over;
}

Control reach this point react after the calling
MACHINE.inputAndReact(EXECID)...

Orchestrate actions taking (logical) time

ABORT (NOW (stop)) {
    SUSPEND TOGGLE (NOW (playPause)) {
        VAR player = new Audio();
        EXEC (player.src = VAL (src),
            player.play(),
            player.addEventListener("ended", function() {
                MACHINE.inputAndReact(EXECID)
            })),
            ONSUSP player.pause()
        ONRES player.play()
        ONKILL player.stop();
        EMIT over;
    }
}

Control reach this point react after the calling
MACHINE.inputAndReact(EXECID)...

... If EXEC is not preempted nor suspended.
Add parallel arms between reactions

```javascript
var m = new hh.ReactiveMachine(
  MODULE {
    FORK {
      LOOP {
        ATOM {
          console.log("The first arm says hello!")
        }
        PAUSE;
      }
    }
    PAR {
      LOOP {
        ATOM {
          console.log("The second arm says hello!")
        }
        PAUSE;
      }
    }
  }
);
```
Add parallel arms **between reactions**

```javascript
var m = new hh.ReactiveMachine(
MODULE {
    FORK foo {
        LOOP {
            ATOM {
                console.log("The first arm says hello!")
            }
            PAUSE;
        }
    }
    PAR {
        LOOP {
            ATOM {
                console.log("The second arm says hello!")
            }
            PAUSE;
        }
    }
}
);
```

Identification of the parallel instruction
Add parallel arms **between reactions**

```javascript
var m = new hh.ReactiveMachine(
  MODULE {
    FORK foo {
      LOOP {
        ATOM {
          console.log("The first arm says hello!")
        }
        PAUSE;
      }
    }
    PAR {
      LOOP {
        ATOM {
          console.log("The second arm says hello!")
        }
        PAUSE;
      }
    }
    m.react();
    var parFoo = m.get("foo"); parFoo.appendChild(
      LOOP {
        ATOM {
          console.log("The third arm says hello!")
        }
        PAUSE;
      }
    );
    m.react();
  }
)
```
Add parallel arms between reactions

```javascript
var m = new hh.ReactiveMachine(
  MODULE {
    FORK foo {
      LOOP {
        ATOM {
          console.log("The first arm says hello!")
        }
        PAUSE;
      }
    }
    PAR {
      LOOP {
        ATOM {
          console.log("The second arm says hello!")
        }
        PAUSE;
      }
    }
    PAR {
      LOOP {
        ATOM {
          console.log("The third arm says hello!")
        }
        PAUSE;
      }
    }
  }
);```
Symbolic and distributed debugger

- Representation of source code with colors according to program state after each reaction.
- Allows to queue reactions and to react step-by-step.
- Controlled remotely.
Symbolic and distributed debugger

- Representation of source code with colors according to program state after each reaction.
- Allows to queue reactions and to react step-by-step.
- Controlled remotely.

Hiphop.js program & debugger

Remote debugger control
Demo!
Thanks you for you attention!

ABORT (NOW (coffee) || NOW (timeout)) {
    LOOPEACH (NOW (question)) {
        RUN (answerQuestionModule);
    }
}

https://github.com/colinvidal/hiphop