Amalgam: Hardware Hacking for Web Developers with Style (Sheets)

Jorge Garza, Devon J. Merrill, Steven Swanson

Department of Computer Science and Engineering,
University of California San Diego, San Diego, CA, USA

NVSL

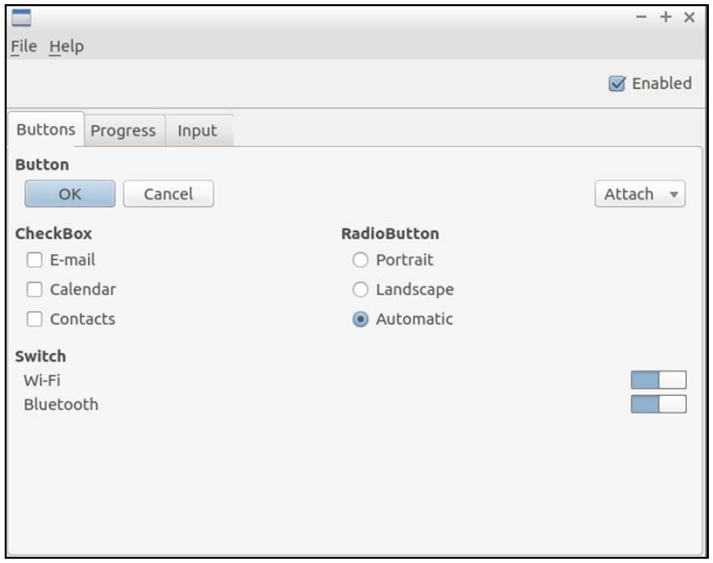
Embedded devices with screens

Could you build one of these devices in 2 day only having web programming skills?

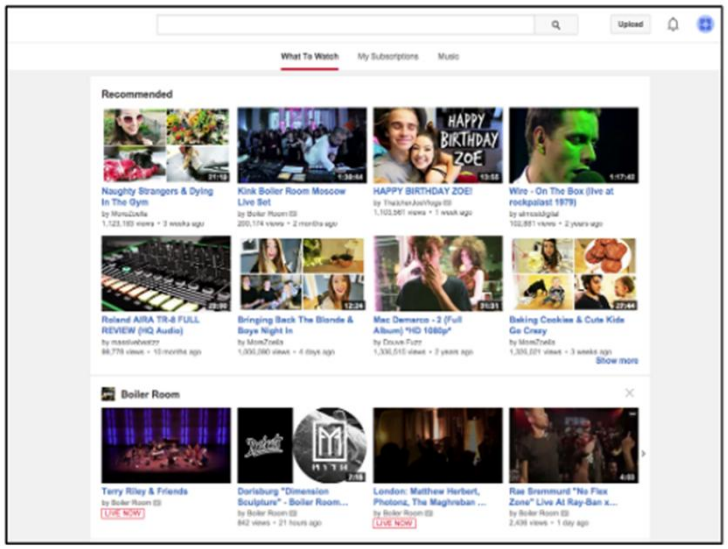
Can we build complex embedded devices with minimal effort?

NVSL

Choosing a GUI framework for your device

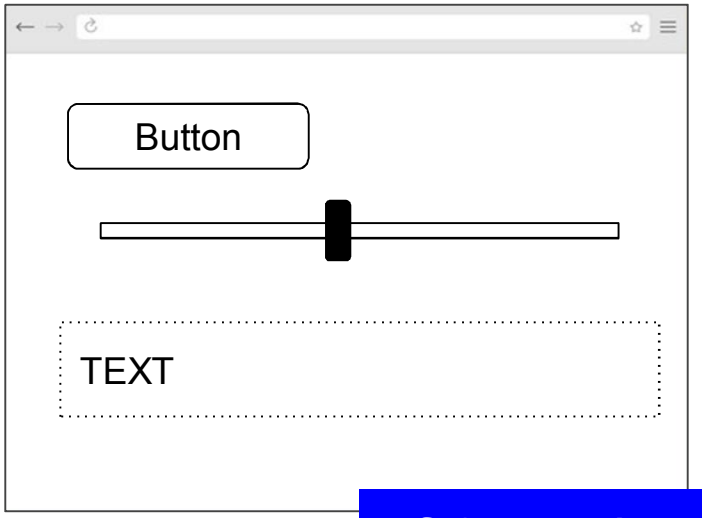


Traditional Desktop GUIs
(e.g., Qt, GTK+, WPF, JavaFX, etc)



Web Programming Technologies
(HTML, Javascript, CSS)

How to Interface between soft and hard components?



Software code ? **Hardware code**

Graphical Interface with software components

HTML/CSS/Javascript

Hardware components

C/C++

Interfacing options

Which one requires the least programming effort?

#	Programming Languages	Interface Option	Programming Effort
1.-	C+JS	**Device Driver** (FILE I/O Operations)	**HIGH**
2.-	C+JS	**Client/Server + Standard Device Drivers**	**MEDIUM**
3.-	Pure JS	**Same Programming Language + Standard Device Drivers**	**LOW**

Interfacing options

Standard device drivers are device drivers that can perform basic hardware communication operations. For example I2C, SPI, Serial and GPIO.

```
$ ls /dev
i2c-1        - I2C
spidev0.0    - SPI
ttyS0        - Serial
```

Advantage: we can implement our hardware code in user space without the need to implement our own device driver.

Interfacing options

Which one requires the least programming effort?

#	Programming Languages	Interface Option	Programming Effort
1.-	C+JS	**Device Driver** (FILE I/O Readings)	**HIGH**
2.-	C+JS	**Client/Server + Standard Device Drivers**	**MEDIUM**
3.-	Pure JS	**Same Programming Language + Standard Device Drivers**	**LOW**

Interfacing options

Which one requires the least programming effort?

#	Programming Languages	Interface Option	Programming Effort
1.-	C+JS	**Device Driver** (FILE I/O Readings)	**HIGH**
2.-	C+JS	**Client/Server + Standard Device Drivers**	**MEDIUM**
3.-	Pure JS	**Same Programming Language + Standard Device Drivers**	**LOW**
4.-	HTML / CSS / JS (Amalgam)	**Deeply Integration of hardware component attributes into the HTML and CSS Syntax + Same Programing Language + Standard Device Drivers.**	**EXTREMLY LOW** HOW?

Amalgam

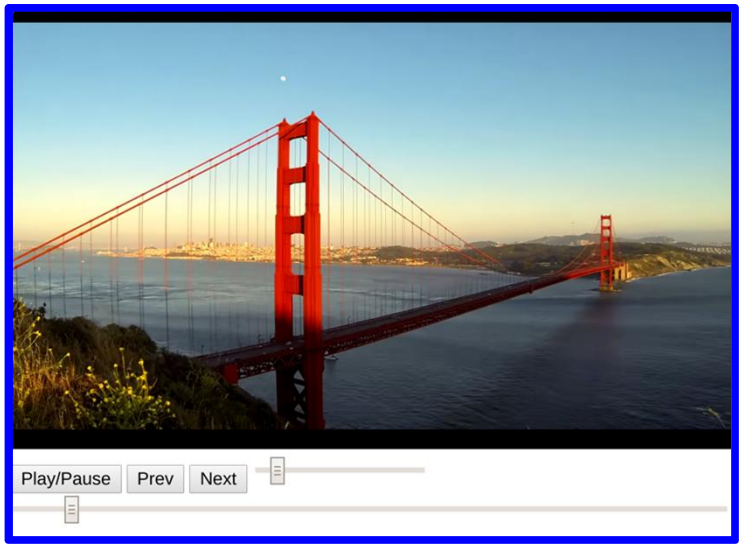
Amalgam

Video Player

Physical Video Player

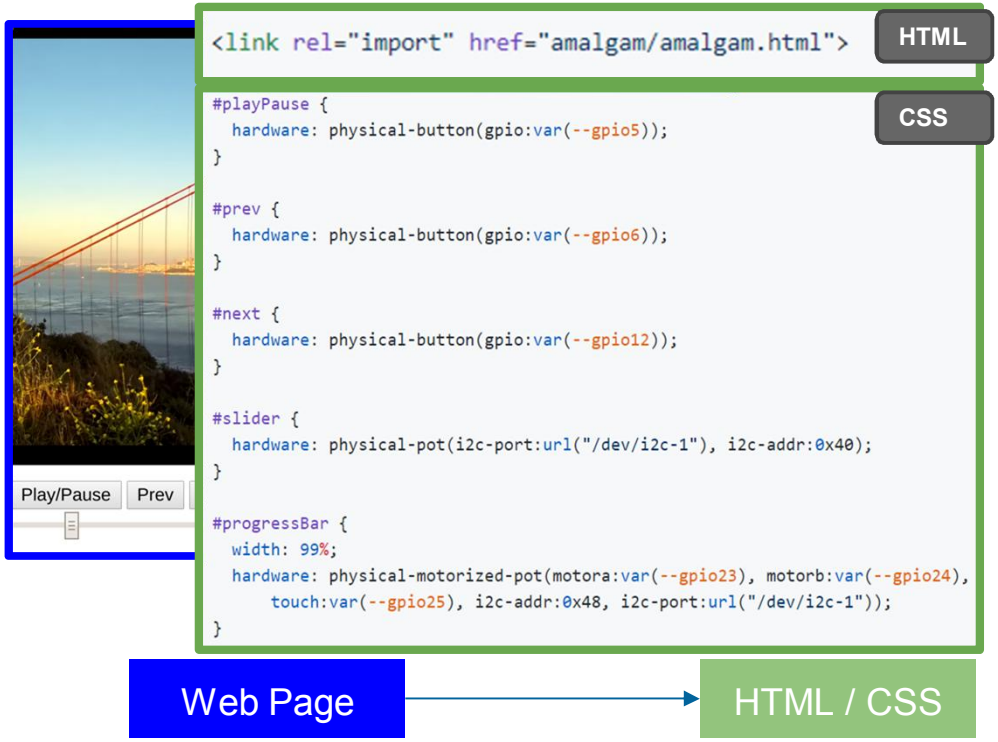
“MAKE DEVICES FROM WEB PAGES”

Making a device with Amalgam



Web Page

Making a device with Amalgam

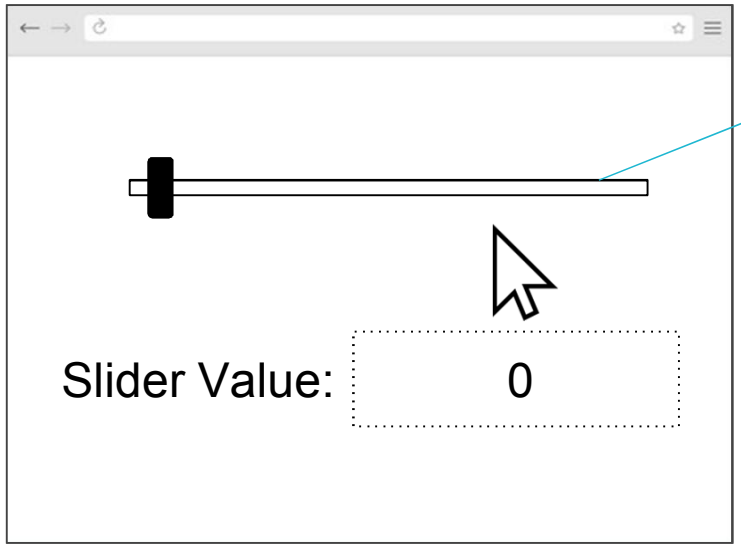


Making a device with Amalgam

How Amalgam Works?
Amalgam internals

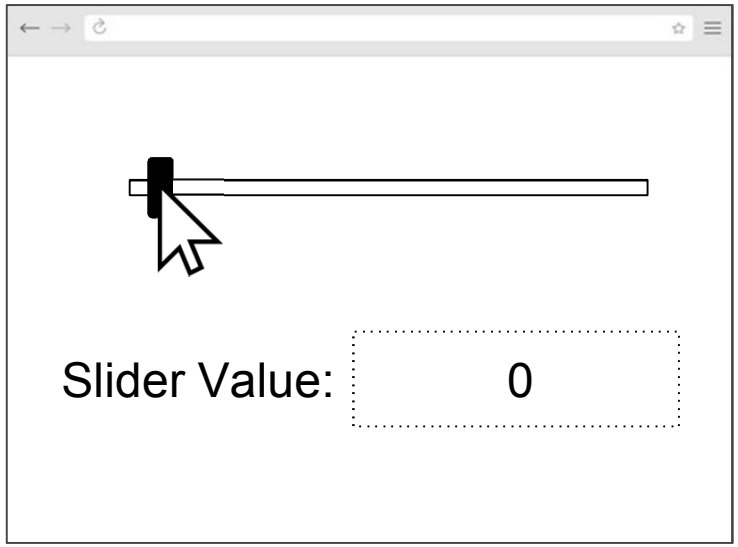
NVSL

Amalgam: a simple example

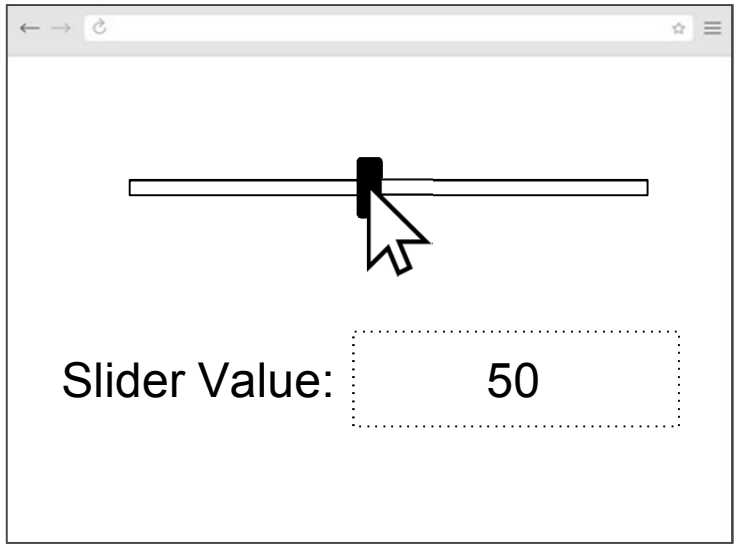


Slider with range from 0 to 100.

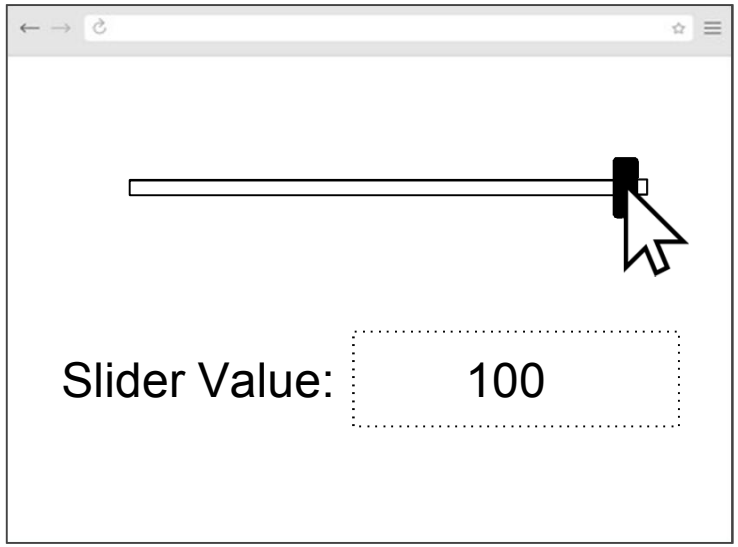
Amalgam: a simple example



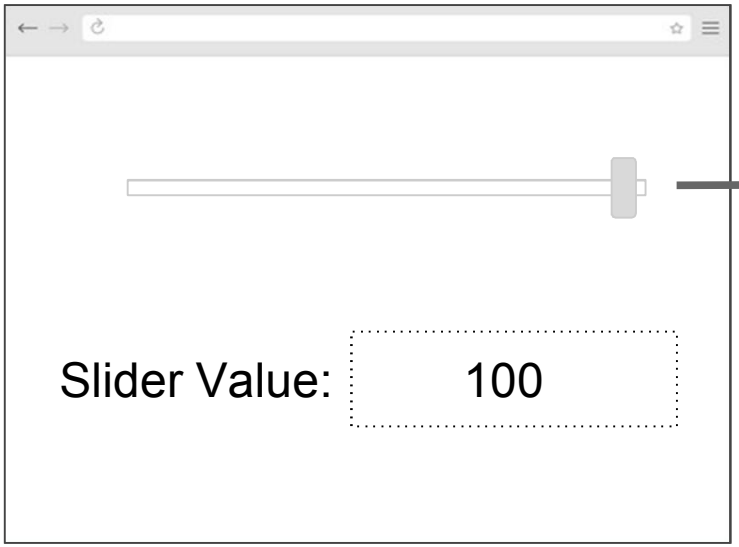
Amalgam: a simple example



Amalgam: a simple example



Amalgam: a simple example

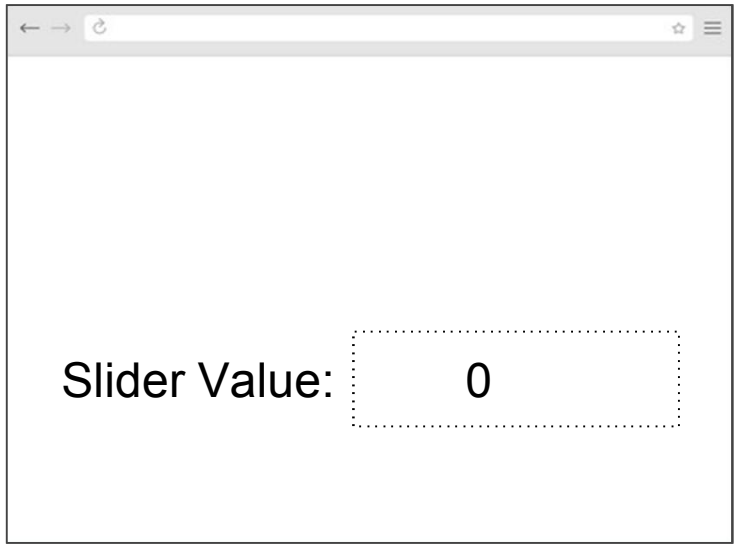


MOVE TO PHYSICAL DOMAIN

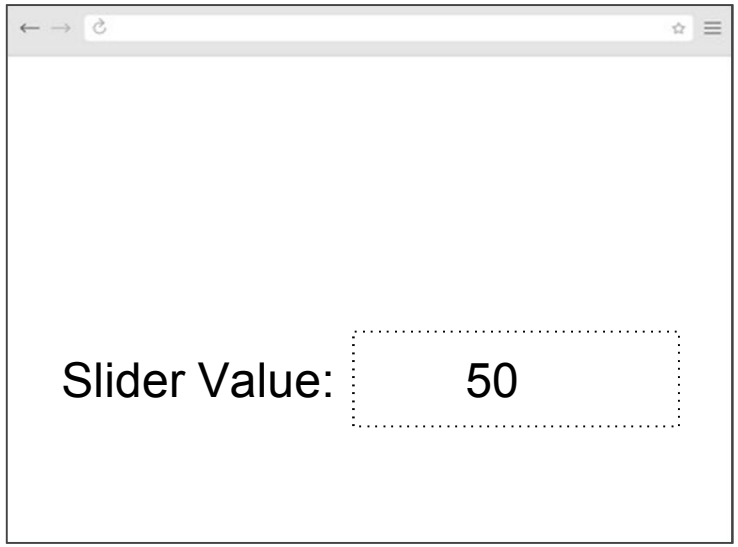
Potentiometer

WE WANT SAME BEHAIVOR!

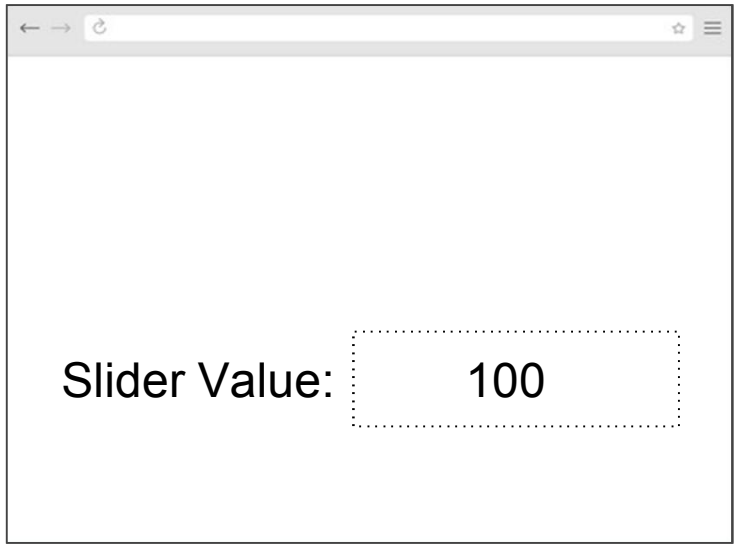
Amalgam: a simple example



Amalgam: a simple example



Amalgam: a simple example



How do we transform the interface of software components to work with hardware components?

Software component
(slider)

HARDENING

Hardware component
(potentiometer)

HARDENING PROCESS

1.- LOOK FOR A HARDWARE COMPONENT WITH SIMILAR BEHAVIOR

2.- ADD AMALGAM ENHANCED CSS

3.- LET THE COMPILATION PROCESS TAKES PLACE

Source: Pending (Published at the International Conference of Web Engineering 2019)

Hardening process

INPUT

a) Soft element HTML

```
<input type="range" min="0" max="100" step="1" id="slider">
```

soft HTML attributes

b) Amalgam-enhanced CSS CSS

```
#slider {
 hardware: physical-pot ( adc-channel: 1, i2c-addr: 0x48,
                          i2c-port: url('/dev/i2c-1') );
}
```

The hardware CSS property

```
#slider {
 hardware: physical-pot ( adc-channel: 1, i2c-addr: 0x48,
                          i2c-port: url('/dev/i2c-1') );
}
```

Attributes required by the Hard element

Name of the Hard element to be used from a list of Hard elements

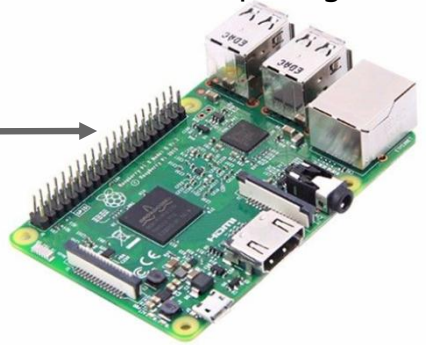
Potentiometer

Analog

Analog to Digital

I2C Standard
Device Driver

RPI Computing Board



Hard Element: A hard element is a web component with hardware properties

Hardening process

INPUT

a) Soft element HTML

```
<input type="range" min="0" max="100" step="1" id="slider">
```

soft HTML attributes

b) Amalgam-enhanced CSS CSS

```
#slider {
 hardware: physical-pot ( adc-channel: 1, i2c-addr: 0x48,
                          i2c-port: url('/dev/i2c-1') );
}
```

Hardening process

INPUT

a) Soft element HTML

```
<input type="range" min="0" max="100" step="1" id="slider">
```

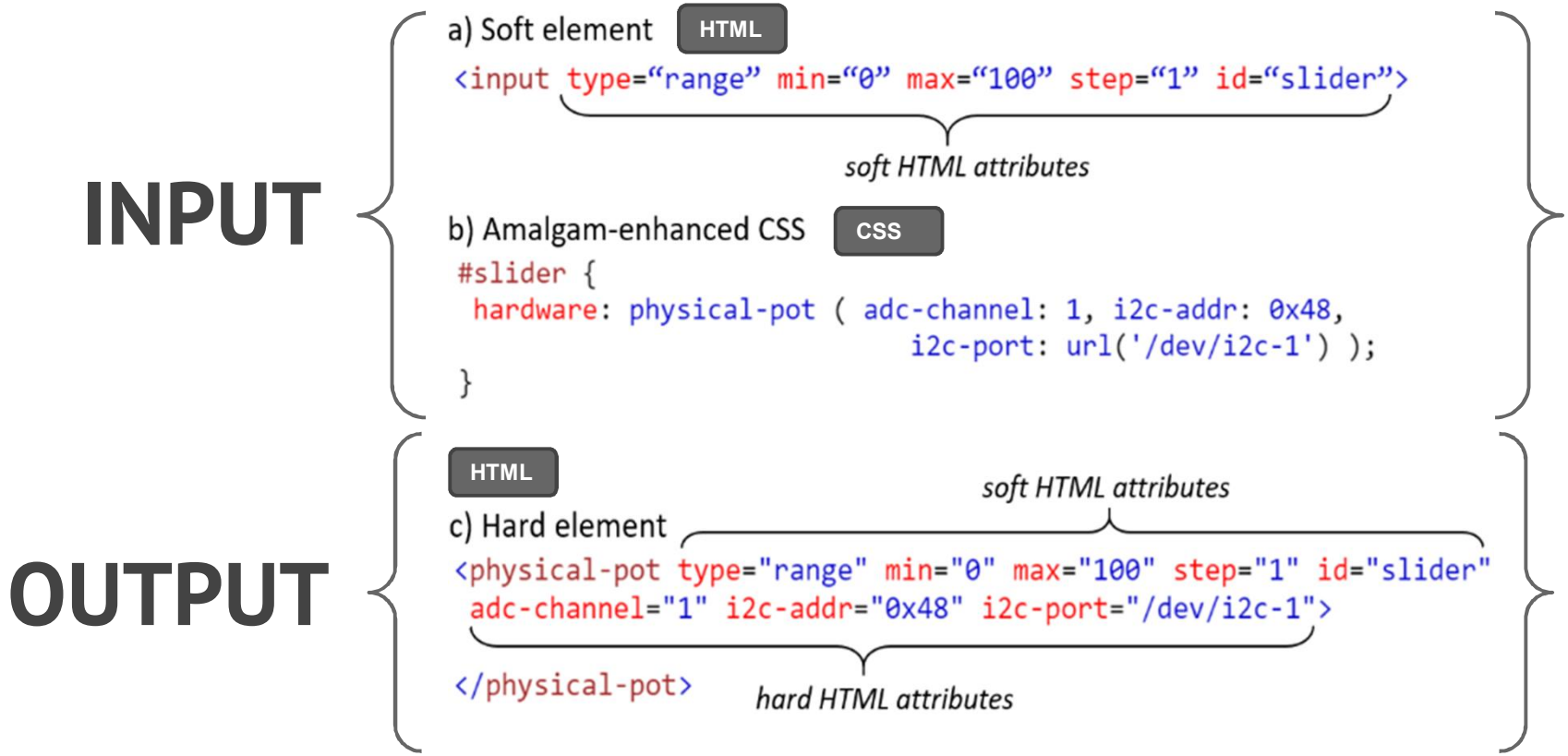
soft HTML attributes

b) Amalgam-enhanced CSS CSS

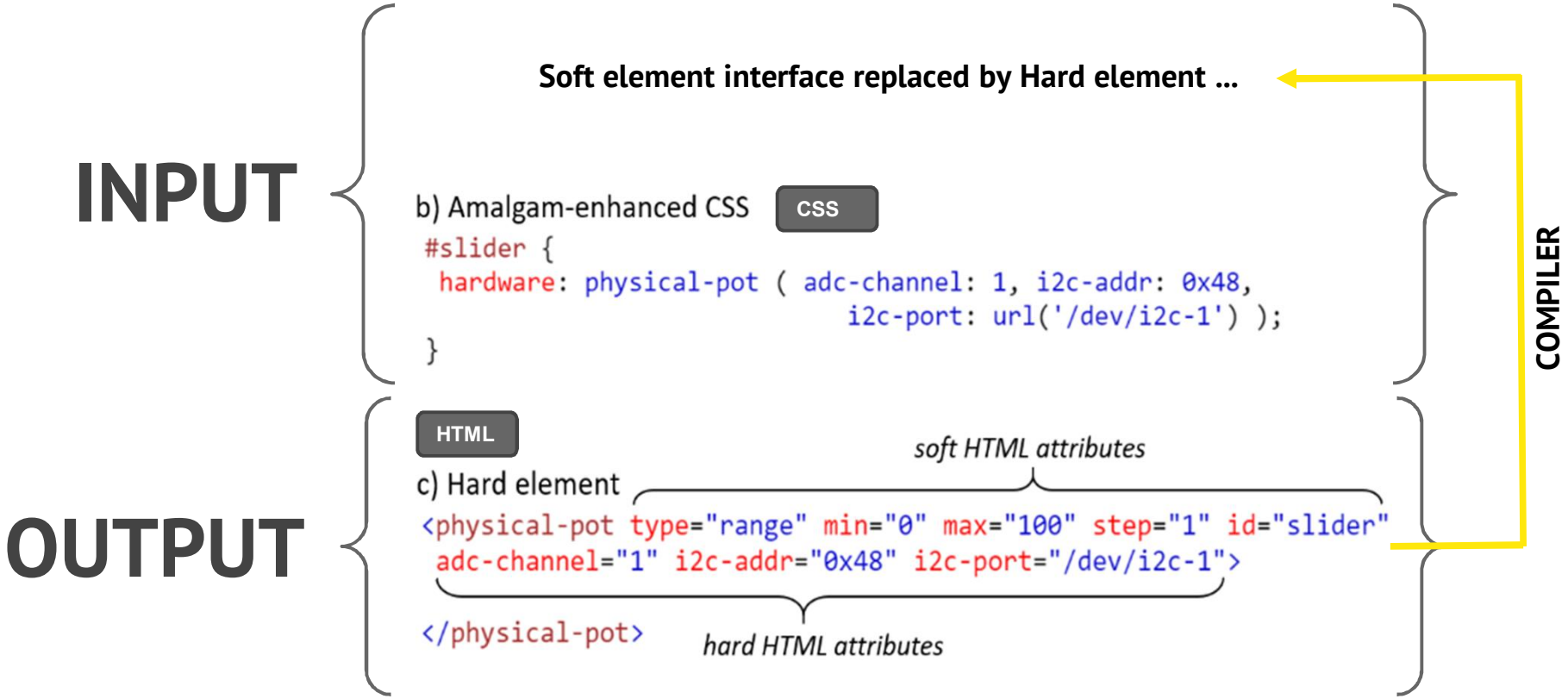
```
#slider {
 hardware: physical-pot ( adc-channel: 1, i2c-addr: 0x48,
                          i2c-port: url('/dev/i2c-1') );
}
```

COMPILATION PROCESS
(AT RUN TIME)

Hardening process



Hardening process



HARD ELEMENT

```
1  class PHYSICAL_BUTTON extends HTMLElement {
2   constructor () { super(); this.gpio; }
3
4    // Monitor attribute changes.
5    static get observedAttributes() {
6      return ['onclick', 'gpio'];
7    }
8
9    connectedCallback() {
10     // Initialize GPIO
11     Linuxduino.pinMode(this.gpio, Linuxduino.INPUT);
12     // Start Reading GPIO
13     setInterval( () => {
14       // Call 'onclick' if physical button pressed
15       if (Linuxduino.digitalRead(this.gpio)  ==
             Linuxduino.HIGH) {
16         this.click();
17       }
18     },200);
19   }
20
21  // Respond to attribute changes.
22  attributeChangedCallback(attr, oldValue, newValue){
23     if (attr == 'gpio') {
24       this.gpio = parseFloat(newValue);
25     }
26  }
27
28 }
29
30 customElements.define('physical-button',
       PHYSICAL_BUTTON);
```

Web component created tag:
<physical-button><\physical-button>

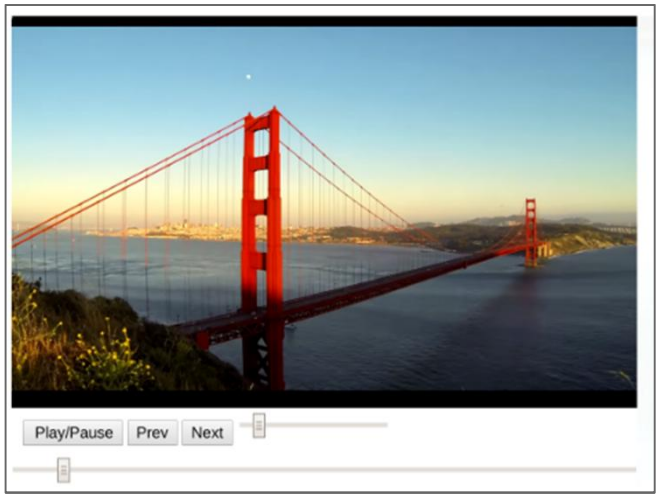
List of Hard elements

Hard element tag	*Amalgam version*	*Compatible soft element tag*	*Notes*
<physical-pot>	Rotary Potentiometer (or "pot")	<input type="range">	Triggers 'oninput' when potentiometer input value is changed.
<physical-encoder>	Rotary encoder	<input type="range">	Triggers 'oninput' when potentiometer input value is changed.
<physical-motorized-pot>	Linear motorized pot	<input type="range">	Triggers 'oninput' when potentiometer input value is changed, also setting the 'value' attribute can set the slider position.
<physical-rgb-led>	RGB LED		Color is set to the CSS background-color property.
<physical-button>	Tactile push-button	<button>	Triggers 'onclick' event at button press.
<physical-servo-motor>	Servo motor		Servo angle is set to angle rotation of CSS transform propery.
<physical-lcd>	LCD text display		Text is set with a hard attribute.
<physical-seven-segments>	LED numerical display		Numbers are set with a hard attribute.
<physical-weight-sensor>	Load cell	<input type="range">	Measured weight is available via Angular ng-bind attribute.

Table 1. Amalgam's hard elements

NYSL

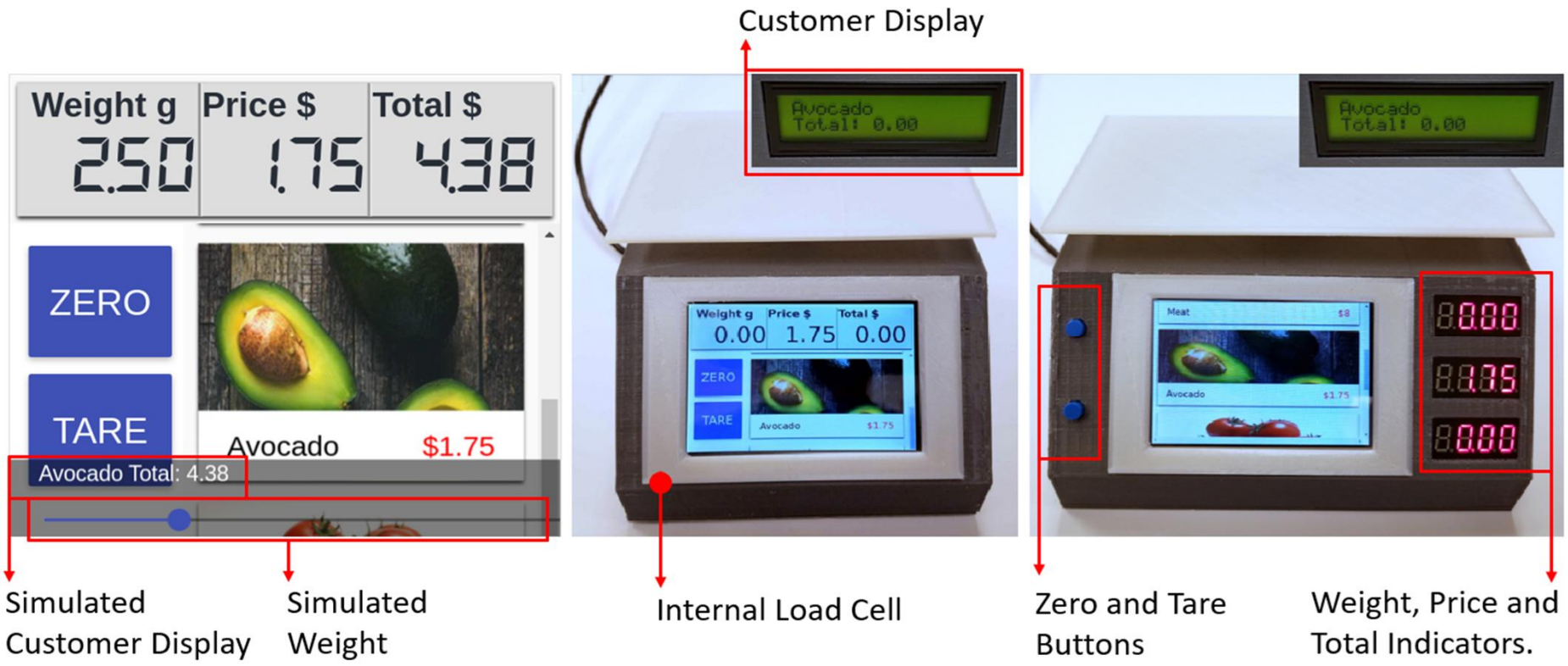
Property 1: Easy Hardware Emulation



=

- Since hard elements have the same interface as soft elements, existing application code requires no modification.
- Therefore, software can be implemented long before the hardware is complete. (Hardware emulation)

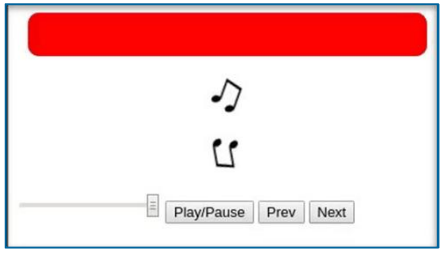
Property 2: Fast Prototyping



Property 3: Hardware can inheritance CSS attributes

```
<link rel="import" href="amalgam/amalgam.html">
...
<body>
<button onclick="playPause()" id="playPause"
  style="hardware: physical-button( gpio: var(--gpio5) )"> playPause
</button>  <!-- Play/Pause button -->
<button onclick="prevSong()" id="prev"
  style="hardware: physical-button( gpio: var(--gpio6) )"> Prev
</button>  <!-- Previous Song button -->
<button onclick="nextSong()" id="next"
  style="hardware: physical-button( gpio: var(--gpio12) )"> Next
</button> <!-- Next Song button -->
<input type="range" min="0" max="1" step="0.1" value="1" id="slider"
  style="hardware: physical-pot( adc-channel: 1, i2c-port: url('
      /dev/i2c-1'),
  i2c-addr: 0x48"> <!-- Volume -->
<div class="rythm color1"
  style="hardware: physical-rgb-led( spi-port: url('/dev/spidev0.0') )">
</div> <!-- RGB LEDs -->
<div class="rythm twist1"
  style="hardware: physical-servo-motor( servo-channel: 0, i2c-port: url
      ('/dev/i2c-1' ), i2c-addr: 0x48 )">
</div> <!-- Servo Motor 1 -->
<div class="rythm twist2"
  style="hardware: physical-servo-motor( servo-channel: 3, i2c-port: url
      ('/dev/i2c-1' ),  i2c-addr: 0x40 )">
</div> <!-- Servo Motor 2 -->
</body>
```

HTML / CSS

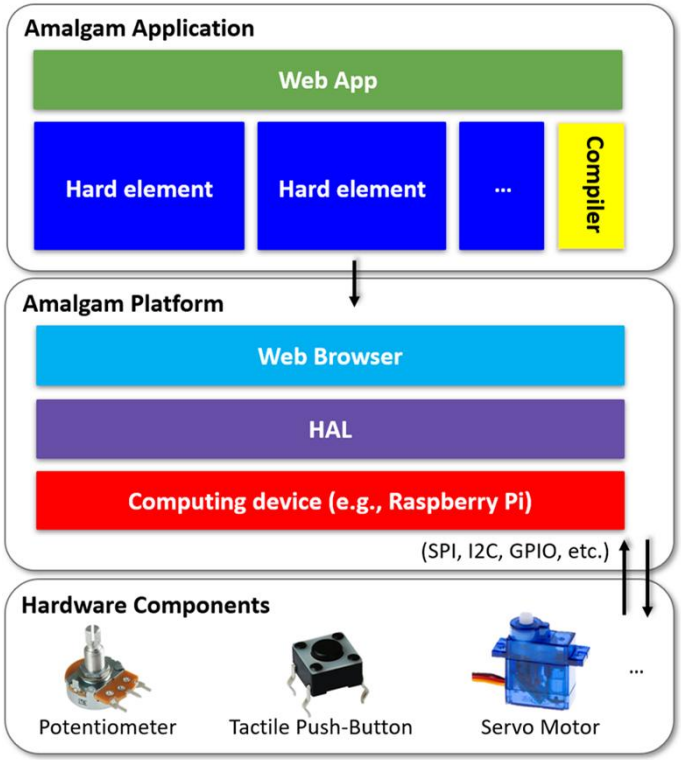


Dancing Speaker

Amalgam architecture overview

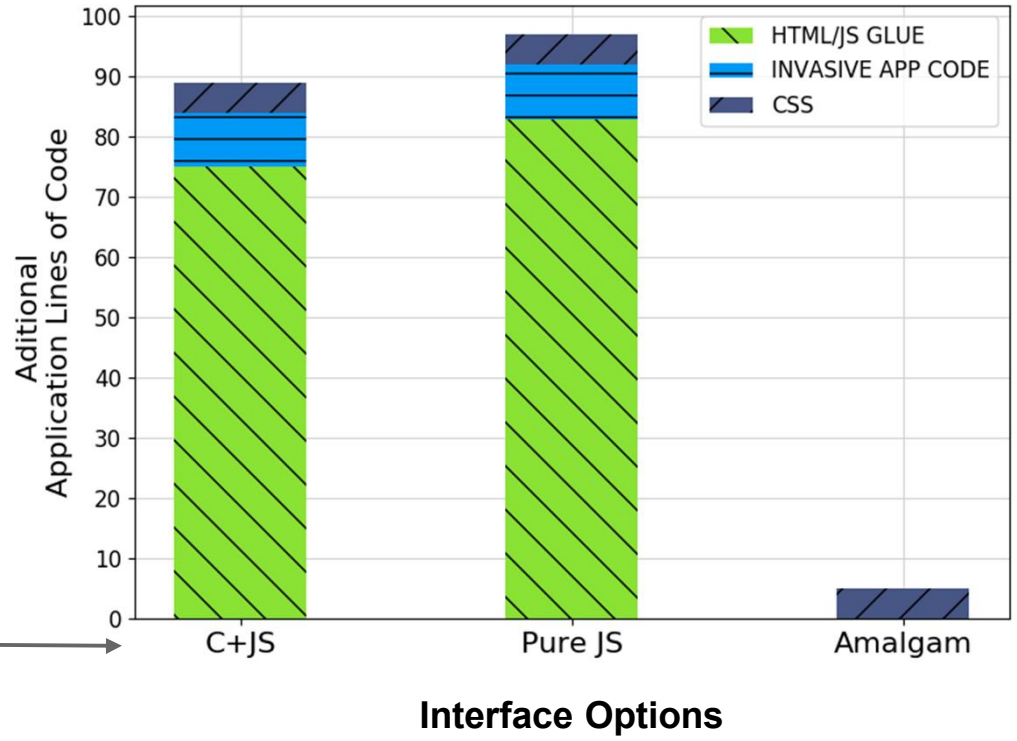
Hosted locally
or in a server

Locally



Results

Programming effort of different interface options compared to Amalgam



Hardware code not included, the same for all.

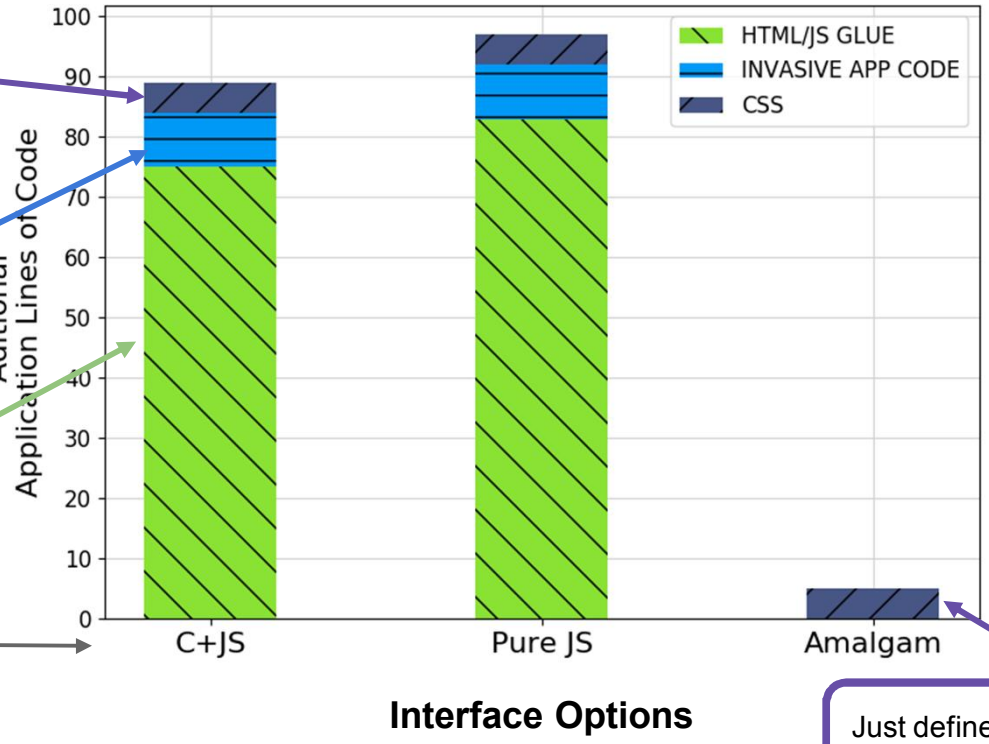
Client/Server code not included

Programming effort of different interface options compared to Amalgam

Remove soft elements manually from the graphical interface

Application code needs to be changed to interact with the hardware code. (e.g., change volume)

Interaction between HTML and JS. Repeated code needed for each similar component (e.g., a button).



Hardware code not included, the same for all.

Client/Server code not included

Just define in CSS which elements to harden.

Q & A

https://github.com/NVSL/amalgam

NVSL