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

Jorge Garza(✉), Devon J. Merrill, and Steven Swanson

Department of Computer Science and Engineering,
University of California San Diego, San Diego, CA, USA
{jgarzagu,djmerrill,swanson}@eng.ucsd.edu

Abstract. Web programming technologies such as HTML, JavaScript, and CSS have become a popular choice for user interface design due to their capabilities: flexible interface, first-class networking, and available libraries. In parallel, driven by the standards set by the mobile companies, embedded devices manufacturers now want to replicate these capabilities. As a result, embedded devices that use web technologies for their graphical interface have started to emerge. However, the programming effort required to integrate web technologies with embedded software hinders its adaption. In this paper, we introduce Amalgam, a system that facilitates the development of embedded devices that use web programming technologies. Amalgam does this by translating the physical interface of embedded hardware components found (e.g., a push button) directly into the HTML and CSS syntax. Our system reduces the programming effort required to develop new embedded devices that use web technologies, as well as adds new interesting capabilities to the design of these. We show Amalgam's capabilities by exploring three embedded devices built using web programming technologies. Also, we demonstrate how Amalgam reduces programming effort by comparing two traditional approaches of building one of these devices against Amalgam. Results show our system reduces the lines of code required to integrate hardware elements into an embedded device application to a line of code per hardware component added to the device.

Keywords: Rapid development · Embedded devices · IoT · Web user interface · CSS · HTML

1 Introduction

With the rise of the Internet of Things (IoT) and smart, connected devices in the home, workplace, and environment, the web and its underlying technologies are pushing up against the real world in a wide range of domains. Connected sensors, web-enabled appliances, and personal electronics all require software that interacts seamlessly with the physical world (e.g., the user or the environment), cloud-based services, and local compute resources. The growing demand for these devices means they need to be easy to program.

Building these smart, connected devices requires programmers to manually bridge the gap between tangible user interfaces (e.g., buttons, knobs, and displays), sensors (e.g., temperature, light, and movement), and actuators (e.g., servos, motors, and lab equipment) and software.

On the software side, web technologies – Javascript, CSS, HTML and the universe of libraries available for them – are state-of-the-art for developing rich user interfaces, provide deep integration network services, and are the languages of choice for a large population of developers.

Web programming technologies provide a clean separation between program logic, interface structure, and appearance. They make it simple to re-style an interface for a new device or adapt an existing interface to a new form factor (such as desktop to mobile). These tools are so powerful that they have become the default user interface design tools for fixed-function mobile devices, desktop applications, and mobile applications.

For hardware, the tools of choice remain C and C++ which can easily handle controlling hardware components (e.g., interrupts, pin assignments, and device drivers). However, they make building user interfaces and networking communication more cumbersome.

Creating a seamless experience that blends on-screen, soft controls, sensors, and actuators is challenging because the elegant separation that HTML, CSS, and JavaScript have does not extend across the hardware/software boundary. In practice, the tools available for hard and soft elements differ in syntax, operation philosophy, and requirements.

This problem is ubiquitous in modern devices. The interfaces to embedded devices – from personal fitness monitors to home appliances – have sophisticated, polished, and powerful user interfaces. Even small devices (e.g., the Apple Watch) typically run full-blown operating systems that can support high-level languages for graphical user interfaces. According to an annual industry survey of embedded designers, 67% of new embedded designs utilize an operating system, and 49% use graphical interfaces [17].

Embedded devices with graphical interfaces can have a blending of *soft* and *hard* components that provide information to or from the user. Examples of soft components can be on-screen buttons, range sliders, and indicators in the form of text or graphics. Hard or physical components include tactile buttons, knobs, and sensors (such as temperature, heart rate, and so forth), actuators (such as servo motors) and hardware indicators (for example, status lights).

Previous attempts to address this problem simply translate the same complex interfaces into higher-level languages, rather than deeply integrating hard elements into the idioms and tools high-level languages provide. This does not solve the problem: Programmers must still treat physical interface components differently than their soft counterparts.

Indeed, several projects [21,23] provide JavaScript libraries for controlling robots [6] and general embedded systems [18], but they leave behind the power of CSS and HTML, preventing deeper integration with existing programming toolkits and tools.

As a result, web programmers that want to build software that deeply integrates software and hardware cannot leverage their own experience; the wealth of training, documentation, and message boards; or the myriad web programming frameworks that are available. Instead, they must develop custom solutions to bridge the gap.

We propose Amalgam, a toolkit that extends web programming technologies across the hardware/software boundary by seamlessly including hardware devices into Javascript, CSS, and HTML. Amalgam exposes the interfaces of hardware components like buttons, sensors, lights, and motor as document object model (DOM) objects with the same interface as their analogous HTML elements (e.g., `<button>`, `<input type=range>`, etc.).

Amalgam lets programmers *harden* conventional DOM objects into hardware device components using a simple CSS directive. The directive controls whether a particular component appears on-screen or as a physical component, and describes how the device connects physically to the computing platform.

As a result, moving a button from on-screen to the real world requires just editing a CSS property and physically connecting the button. Application logic does not change because the interface remains the same. More important, existing frameworks like Angular [1] and JQuery [15] work just as well with hard elements as soft.

This paper makes the following contribution: By integrating hardware component interfaces into the web user interface syntax, our system allows for rapid development and prototyping of complex embedded devices. To the best of our knowledge, this is the first work that explores the integration of hardware interfaces directly into the HTML and CSS syntax. Furthermore, with our system, extra capabilities are observed.

For instance, hardware components can inherit CSS capabilities. For example, the programmer can create complex lighting effects by using CSS to animate the color of an RGB LED. Likewise, setting the HTML content of a `<span>` that has been hardened into a display can change the contents of the display.

We have implemented Amalgam as a Javascript framework and developed a small but useful library of hardware components. We demonstrate Amalgam's capabilities by using these components to create three embedded devices with rich hardware/software interfaces. We demonstrate that Amalgam works seamlessly with existing web-programming frameworks and libraries to build complex, responsive interfaces for these devices. We also describe Amalgam's implementation and measure how Amalgam makes it easier to develop these kinds of devices.

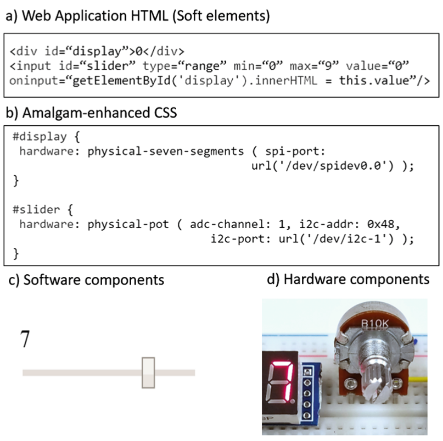
The rest of this paper is organized as follows. Section 2 gives a overview of Amalgam, and Sect. 3 illustrates Amalgam's capabilities by describing three Amalgam devices. Section 4 evaluates the impact of Amalgam on developer effort. Finally, Sect. 5 describes related work, and Sect. 6 presents our conclusions.

2 Amalgam

Amalgam is a Web API that integrates hardware components into web programming tools in a natural and transparent way. It provides a new style attribute (`hardware`) that convert on-screen elements of web-based interfaces into a hardware device. We call this process *hardening* the on-screen element. Hardening allows, for instance, the replacement of an on-screen button with a physical button. The element's interface remains the same, so the application software does not need to change.

We have implemented Amalgam as a JavaScript library. It leverages Web Components [11] and Web Assembly [13] to build DOM elements that interface with hardware, and it provides a simple compiler that parses a web page's CSS style sheets and hardens elements according the `hardware` directives it finds.

This section describes Amalgam's programming interface, presents a simple example of Amalgam in action, describes the library of physical components we have implemented, and what is required to create a new one.


a) Web Application HTML (Soft elements)

```
<div id="display">0</div>
<input id="slider" type="range" min="0" max="9" value="0"
oninput="getElementById('display').innerHTML = this.value"/>
```

b) Amalgam-enhanced CSS

```
#display {
 hardware: physical-seven-segments ( spi-port:
                                     url('/dev/spidev0.0') );
}

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

Fig. 1. Styling hardware with Amalgam (a) Describes the interface for a simple numerical display controlled by a slider. Applying an Amalgam-enhanced CSS style sheet (b), produces the same on-screen version of the interface (c) implemented in hardware (d).

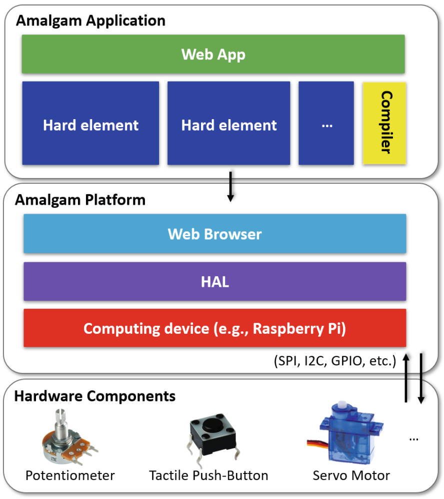


Fig. 2. Amalgam Platform: An Amalgam application, a web app that includes the Amalgam Web API, can run on Amalgam Platforms which includes a web browser engine that can communicate with hardware through the HAL.

2.1 Overview

Amalgam's programming interface is simple by design. It lets programmers convert existing DOM elements, which we call *soft elements*, into hardened elements that exist in the real world.

Figure 1 shows a simple Amalgam application. In the figure, (a) shows the HTML for a range `<input>` and a `<div>` along with the event callbacks to ensure that the `<div>` displays the value of the `<input>`. (c) depicts the web page in browser running on a Raspberry Pi [9].

The CSS code in (b) hardens both elements by setting their `hardware` attribute. It converts the `<div>` into a seven-segment LED display and the `<input>` into a rotary knob potentiometer. The photo (d) shows the hardware. Turning the knob updates the display. No other changes are necessary to the code.

The value of the `hardware` attribute describes what kind of hardware to use (in this case, the knob and the display) and how the two components connect to the Raspberry Pi. In this case, the display connects via SPI and the potentiometer connects to the first channel of an analog-to-digital (ADC) integrated circuit connected via I2C

Once hardened, the components continue to behave just like the original soft components. It has the same DOM methods and emits the same events (e.g., when the knob moves, the `<input>` emits `onchange`). If the hardened component has a display capability (e.g., an RGB LED), the programmer can style or animate it with CSS.

Since hard elements have the same interface as soft elements, existing application code requires no modification. In particular, web programming frameworks function as expected without any changes. Cleanly integrating hardware components into web programming technologies offers multiple benefits.

- **Easy Hardware Emulation.** Amalgam decouples application design from hardware design. Software developers can implement the software for a soft version of a device long before the hardware is complete.
- **Faster Design Iteration.** Developers can rapidly explore different designs by hardening different parts of a user interface without needing to modify the application logic.
- **Faster Development.** Amalgam lets developers leverage the universe of available JavaScript frameworks to quickly build complex applications.
- **Automatic Web Integration.** Because they are web applications, Amalgam applications have first-class access to web services, the cloud, etc.

Amalgam makes it easy for programmers to control hardware devices, but they still need to assemble the device. Moreover, they need some familiarity with the hardware and its limitations. Amalgam can replace any HTML element with any hard component, but the programmer must be aware of potential limitations. For example, setting the `value` attribute for `<input>`, used for setting the slider position, when hardened into a normal potentiometer it will not move the potentiometer. If the programmer needs that capability, he should use a rotary encoder or a motorized potentiometer. Likewise, the designer must be aware of which pins connect the hardware components, and if those connections change, the programmer must update the CSS.

2.2 Amalgam Platforms

Amalgam applications run an Amalgam *platform*. A platform includes a computing device (e.g., a RaspberryPi), a web programming runtime that includes JavaScript, DOM, and CSS (e.g., a web browser) that the applications run in and a *hardware abstraction layer (HAL)* that provides low-level access to hardware. A web server (running on the platform or remotely) serves the application. Figure 2 shows the components of an Amalgam platform and their relationships to an Amalgam application.

The HAL exposes a standard software interface (i.e., function calls) to common hardware interfaces (i.e., electrical connections to the platform hardware). For the HAL in Amalgam we implemented a version of the Arduino Reference Language [2] for Linux, we call it Linuxduino [7]. Linuxduino accesses low-level hardware through Linux's standard drivers, so it should be portable across the many Linux-based embedded systems that are available. Since it is Arduino-compatible, it supports a huge array of hardware components. Linuxduino was implemented in C++ and compiled to web assembly, so it runs directly in the JavaScript runtime.

The main difference between different Linux-based platforms is the set of electrical interfaces they provide. For instance, Raspberry PI provides 26 digital IO pins, two I2C interfaces, and two SPI interfaces, but no analog inputs or outputs. In addition, the platforms use different naming schemes for their pins.

These differences are visible to Amalgam programmers so moving an Amalgam program between platforms requires adjustments to the CSS that hardens the components. Likewise, if the hardware designer changes which pins connect a particular device to the platform, the CSS must change as well.

2.3 The Amalgam Library

To explore Amalgam's ability to accelerate the design of complex hard/soft interfaces, we built seven hard elements that match existing soft elements HTML interfaces. Table 1 summarizes the elements Amalgam currently supports.

The range of possible hard elements is broader than the set of elements that HTML provides, because many different hardware devices can replace a single HTML element, and even similar hardware devices may connect to the system through different interfaces.

For instance, the example in Fig. 1 hardened the `<input>` into a potentiometer (or "pot") that connected via one ADC channel. It could have instead used a "rotary encoder" that connects via two digital IO lines or a "motorized slide pot" that requires an ADC line and three digital IO lines, two lines to control the motor direction and one to get the user slider touch feedback.

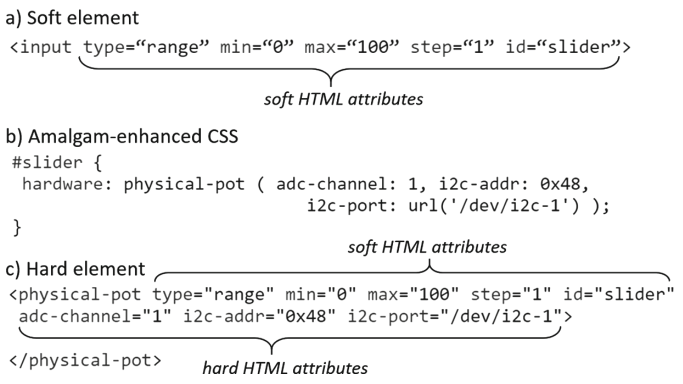
The Amalgam library has entries for each of these alternatives. Their internal software implementations are quite different despite appearing the same to the application (i.e., as a range `<input>`).

Adding new hard elements to Amalgam requires two steps. The first is creating a web component that will interface with the hardware device. The component encapsulates the firmware (written in JavaScript) that controls the hardware via the HAL.

For example, Listing 1.1 is a class that implements a hard button by extending `HTMLElement` and providing three methods: `get_observedAttributes()` defines the attributes this element supports. Whenever an attribute changes, including when an element is hardened, `attributeChangedCallback()` runs and gets updated. Finally, the runtime invokes `connectedCallback()` once after the attributes are updated, indicating that the web component is ready. Here initialization and configuration of hardware is carried out. In this case, a given GPIO number, set in the GPIO attribute value, is configured as input and is physically connected to a button. After that, another function sets up a call back that polls the IO pin every 200 ms, which is enough to detect a button press, and emulates a click when it detects a physical button press (lines 13–18). For this example, the GPIO number can only be initialized once but it works for our embedded device prototypes requirements.

```
 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 customElements.define('physical-button',PHYSICAL_BUTTON);
```

Listing 1.1. A hard element button code example which consist of a typical web component code plus calls to hardware using the Linuxduino HAL library.



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

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

```
<physical-pot type="range" min="0" max="100" step="1" id="slider"
 adc-channel="1" i2c-addr="0x48" i2c-port="/dev/i2c-1">
</physical-pot>
```

Fig. 3. Amalgam compiler hardening of soft elements, (a) Shows a soft element selected by an Amalgam-enhanced CSS property in (b). After compilation (c) shows the hard element HTML which replaces (a).

Table 1. Amalgam's 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	<div>	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	<div>	Servo angle is set to angle rotation of CSS transform property
<physical-lcd>	LCD text display	<span>	Text is set with a hard attribute
<physical-seven-segments>	LED numerical display	<spa>	Numbers are set with a hard attribute
<physical-weight-sensor>	Load cell	<input type="range">	Measured weight is available via Angular ng-bind attribute

The final step is to register the new class with Amalgam so the programmer can use it to harden elements. The code in Listing 1.1 defines a new HTML tag called `<physical-button>` that the programmer can use directly (e.g., `<physical-button onclick="foo()" gpio="1"></physical-button>`) to create a hard button. The registration process makes the Amalgam CSS compiler aware of the class so it can replace an existing tag (e.g., a `<button>`) with a `<physical-button>`.

2.4 Amalgam-Enhanced CSS Style

The `hardware` CSS style attribute controls if and how Amalgam hardens a DOM element. The value of `hardware` describes which hard component should replace the software component and describes which electrical interfaces the corresponding hardware device will connect to.

Figure 3 exemplifies the Amalgam compiler. There (b) shows the CSS code required to harden a soft element (a). Each value for `hardware` starts with the name of the hard component that Amalgam will use to replace the soft element. The remaining arguments are of the form *`attr`*`(`*`value`*`)` that Amalgam uses to set the hard HTML attributes on the hard element (c) it creates. The hard element as well will inherit soft HTML attributes from the soft element in (a) to keep the same web application functionality without any changes. Our prototype implementation uses a JavaScript CSS processor to scan a pages style sheets for `hardware` declarations, and then harden the elements appropriately.

3 Examples

To demonstrate Amalgam, we built three devices[1]: A video player, an commercial food scale, and a dancing speaker. Each device started with an on-screen, soft prototype. We hardened some of the soft components and built a physical prototype of the device. The Amalgam platform used is Raspberry Pi running Linux and Electron [3]. Electron is used to allow web applications to access the file system and hardware through Node.js [8].

3.1 Video Player

The video player appliance (Fig. 4) demonstrates Amalgam's ability to transform an existing web page into the firmware for a physical device. The left side of the figure shows the soft video player built with the Youtube Player API [14]. It provides a familiar on-screen interface for playing videos, including the slider that both displays and controls the playhead location.

[1] The code is available at https://github.com/NVSL/amalgam.

The right side of the figure shows the appliance we built. Videos appear on the screen, but all the rest of the interface is hard. The only difference in software the CSS directives to hardened the three buttons, the volume control, and slider (Listing 1.2). The appliance mimics all the behavior of the original, including the progress bar. We hardened it into a motorized potentiometer that both the software and the user can actuate.

3.2 Commercial Scale

The scale appliance (Fig. 5) shows Amalgam's ability to simplify and accelerated prototyping iterations. The scale has two users: the customer and the salesperson. The salesperson can select products from an illustrated list, see the price per pound, weigh the item, and adjust the scale by zeroing it or setting a tare weight (to account for the weight of a container), and show the total. The customer can see the item's name and the total price displayed on a second display on the reverse side.

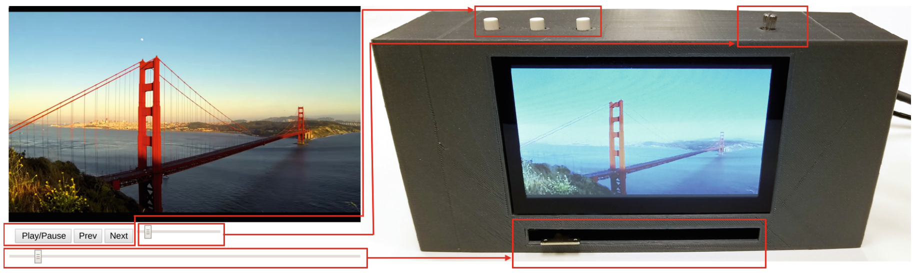


Fig. 4. Video Player: At left, a demo of a video player which provides a familiar on-screen interface for playing videos. At right, Amalgam allows the same demo to drive a fully-tactile interface, the only difference being the style sheet.

The soft version (at left) implements the application logic and these interfaces and via soft elements. The developer can perfect the application logic (including varying the weight on the virtual scale) in a web browser without access to any hardware. The hardware prototype in the center provides a completely soft salesperson interface. While the one at right uses hard components for the buttons and numeric displays. Both of them have a hard customer display. The only software difference between all three versions is a few lines of CSS.

We implemented the soft version using Angular [1], a sophisticated model-view-controller library. Angular makes it trivial to "bind" the output of the load cell to the weight display. Since Amalgam's hard elements have the same interface as normal DOM elements, this works just as easily with hard elements.

```
<link rel="import" href="amalgam/amalgam.html">
...
<!-- Soft elements -->
<body>
<button onclick="playPause()" id="playPause">Play/Pause</button>
<input type="range" min="0" max="10" step="1" value="0" id="progressBar">
</body>
<!-- Amalgam-enchanced CSS -->
<style>
#playPause {
  hardware: physical-button(gpio:var(--gpio5));
}
#progressBar {
  hardware: physical-motorized-pot (motora:var(--gpio23), motorb:var(
      --gpio24),
  touch:var(--gpio25), adc-channel: 2, i2c-addr: 0x48, i2c-port:url("
      /dev/i2c-1"));
}
</style>
```

Listing 1.2. Video Player Code. At the top we show only two of the software components of the Video Player web application, the play-pause button and the progress bar. At the bottom the Amalgam-enhanced CSS required to harden the software elements, process that is carried out by the compiler at run-time.

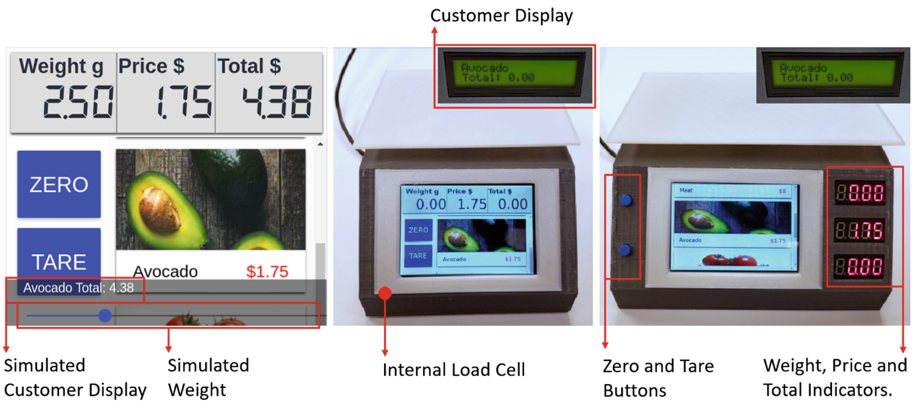


Fig. 5. Evolving Scale: Amalgam allows a spectrum of different implementations with minimal developer effort. From left to right: a software-only mock-up includes a virtual, on-screen load cell and supports software development; a "soft" interface version that uses a touch screen for the main screen and buttons and an LCD for the rear screen; and hybrid version that uses 7-segment displays and tactile buttons.

3.3 Dancing Speaker

Our dancing speaker is a simple demo that (Fig. 6) highlights the power of CSS animations to control hardware. The speaker plays and "dances" to music by waving its arms and flashing lights in time to music. It is a fanciful design that a "maker" might assemble as a hobby project.

Assembling the hardware for the dancing elements is simple, but writing the software for control (e.g., beat detection and complex coordinated transitions) is complex. Instead of writing that code from scratch, our design leverages an unmodified, third-party library called Rythm.js [10] that can make any website dance in time to the music. Rythm.js uses CSS classes (e.g. "rythm twist1") to represent background color and angle rotation changes to `<div>` tags. Applying those classes to the servos and LEDs makes them dance just as well. Listing 1.3 shows the HTML and CSS implementation of our dancing speaker.

4 Impact on Development Time

Amalgam's goal is to make it easier for developers to build physical devices with rich interfaces. To quantify its effectiveness, we built two other versions of the video player: One using pure JavaScript and another using JavaScript and C.

The "C+JS" version uses a simple server implemented in C that exposes hardware components via a TCP socket. The JavaScript that implements the application logic communicates with it via TCP sockets. The "Pure JS" version calls the HAL directly to control the hardware and implements the same functionality that Amalgam's hard elements implement internally. We refer to this as *glue code.*

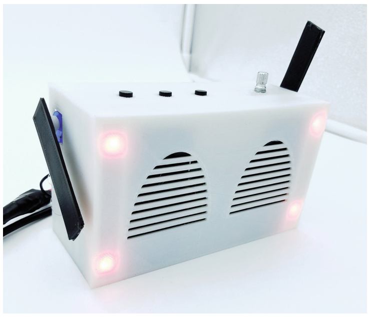


Fig. 6. Our dancing speaker

```
<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>
```

Listing 1.3. Dancing speaker code implementation using Amalgam-enhanced CSS.

Figure 7 compares the lines of code (LOC) required to integrate the hardware components into each version of the application. The measurements do not include the frameworks, libraries, or the server and communication code for C+JS. We also include the lines of code added or changed in the application code to accommodate the change from soft element to hard elements (labeled as “invasive” changes).

The figure shows that Amalgam vastly reduces the effort required to harden components: five lines of CSS in one file compared to over eighty lines of JavaScript and CSS spread throughout the application for Pure JS and C+JS. Amalgam avoids invasive changes completely.

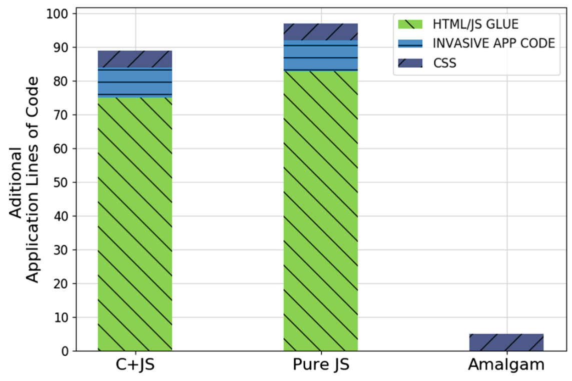


Fig. 7. Programming Effort: Deeply integration of hardware components interfaces into the web languages allows Amalgam to reduce the lines of code needed to integrate these components into web application based electronic devices, therefore reducing development time.

5 Related Work

Amalgam seamlessly integrates hardware components into HTML, CSS, and JavaScript to reduce development effort and facilitate faster prototyping. Below, we place Amalgam in context with other projects with similar goals.

5.1 Integration of Hardware to Web Technologies

Several previous projects have focused on the integration of hardware to web technologies. In particular, the Web of Things [20], and IoT protocols such as MQTT [22] and SOAP [16]. These IoT protocols use web programming technologies (e.g., HTTP, Web Sockets, XML, etc.) to interface remotely with hardware devices which have integrated sensors and actuators. As hardware devices become more powerful at a reduced cost [24] embedded developers are looking to use web programming technologies to also interface locally with hardware. Related efforts adapt JavaScript to run on constrained devices (e.g JerryScript [18]).

Web Browsers have become an extensively used platform that can run across heterogeneous hardware and software platforms, and they provide access to a limited number of hardware components like cameras and microphones [5] via standardized JavaScript APIs.

As web technologies are becoming popular on embedded, mobile devices, other standards for interfacing with hardware components have been included, such as Bluetooth low energy [12] and sensors like accelerometers, gyroscopes and ambient light detection [4]. Still, web browsers standards have not been able to keep up with the myriads of hardware components currently available.

Developers who want to use non-standard (or less common) hardware components with web technologies must do so in an ad hoc manner by developing custom communication protocols or “glue” libraries to provide access in JavaScript. Projects like Jhonny-Five [6] provide these facilities for some hardware devices, but it does not integrate cleanly CSS or HTML. It also does not provide easy access to generic interfaces like I2C and SPI, limiting its generality.

5.2 Rapid Development of Embedded Devices

Many tools exist for the rapid software development of embedded devices. The Arduino Language [2], minimizes the time to develop of embedded software on microcontroller platforms by hiding their low level complexity behind a simple library. TinyLink [19] reduces the lines of code by providing tools that generate the underlying hardware interfaces and binaries required for a target platform. Microsoft .NET Gadgeteer [25] uses a modular hardware platform that is deeply integrated into the Microsoft Visual Studio IDE. Gadgeteer provides hardware abstraction libraries for each supported module and facilitate development by using C# as its main programming language.

Amalgam is similar in some respects to both Arduino and the software support in Gadgeteer: All three projects aim to integrate hardware support into the host language (C for Arduino, C# for Gadgeteer, and Javascript/CSS/HTML for Amalgam). Amalgam, however, improves on the usability of the others by leveraging the flexibility and power of web programming technologies.

6 Conclusions

In this paper, we present Amalgam, a toolkit that deeply integrates hardware devices into web programming technologies. Amalgam enables rapid development and more flexible design iteration for embedded devices. Amalgam lets developers replace soft interface components with hardware components just by changing a CSS file. We implemented Amalgam and evaluated its capabilities by prototyping three devices in a web browser and then "hardening" them into standalone devices. Our results show that Amalgam can significantly reduce the programmer effort required to implement the software for electronic devices.

References

1. AngularJS - Superheroic JavaScript MVW Framework. https://angularjs.org/
2. Arduino Reference. https://www.arduino.cc/reference/en/
3. Electron—Build Cross Platform Desktop Apps with JavaScript, HTML, and CSS. https://electronjs.org/
4. Generic Sensor API. https://www.w3.org/TR/generic-sensor
5. HTML Media Capture. https://www.w3.org/TR/html-media-capture/
6. Johnny-Five: The JavaScript Robotics & IoT Platform. http://johnny-five.io/
7. Linuxduino - A JavScript Library for Communicating with Hardware in a Arduino Style Programming for Any Linux Platform. http://www.w3.org/TR/html5
8. Node.js. https://nodejs.org/en/
9. Raspberry Pi - Teach, Learn, and Make with Raspberry Pi. https://www.raspberrypi.org/
10. Rythm.js - GitHub Pages. https://okazari.github.io/Rythm.js/
11. Specifications - webcomponents.org. https://www.webcomponents.org/specs
12. Web Bluetooth Community Group. www.w3.org/community/web-bluetooth
13. WebAssembly. https://webassembly.org/
14. YouTube Player API Reference for iframe Embeds - Google Developers. https://developers.google.com/youtube/iframe_api_reference
15. Volder, K.: JQuery: a generic code browser with a declarative configuration language. In: Van Hentenryck, P. (ed.) PADL 2006. LNCS, vol. 3819, pp. 88–102. Springer, Heidelberg (2005). https://doi.org/10.1007/11603023_7
16. Dürkop, L., Imtiaz, J., Trsek, H., Jasperneite, J.: Service-oriented architecture for the autoconfiguration of real-time ethernet systems. In: 3rd Annual Colloquium Communication in Automation (KommA) (2012)
17. EETimes: 2017 Embedded Markets Study: Integrating IoT and Advanced Technology Designs, Application Development Processing Environments, April 2017. https://m.eet.com/media/1246048/2017-embedded-market-study.pdf

18. Gavrin, E., Lee, S.J., Ayrapetyan, R., Shitov, A.: Ultra lightweight JavaScript engine for Internet of Things. In: Companion Proceedings of the 2015 ACM SIGPLAN International Conference on Systems, Programming, Languages and Applications: Software for Humanity, SPLASH Companion 2015, pp. 19–20. ACM, New York (2015). https://doi.org/10.1145/2814189.2816270. http://doi.acm.org/10.1145/2814189.2816270
19. Guan, G., Dong, W., Gao, Y., Fu, K., Cheng, Z.: TinyLink: a holistic system for rapid development of IoT applications. In: Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking, pp. 383–395. ACM (2017)
20. Guinard, D., Trifa, V.: Towards the Web of Things: web mashups for embedded devices. In: Workshop on Mashups, Enterprise Mashups and Lightweight Composition on the Web (MEM 2009), in Proceedings of WWW (International World Wide Web Conferences), Madrid, vol. 15 (2009)
21. Kuc, R., Jackson, E.W., Kuc, A.: Teaching introductory autonomous robotics with JavaScript simulations and actual robots. IEEE Trans. Educ. **47**(1), 74–82 (2004)
22. Locke, D.: MQ Telemetry Transport (MQTT) v3. 1 Protocol Specification. IBM Developer Works Technical Library (2010)
23. Osentoski, S., Jay, G., Crick, C., Pitzer, B., DuHadway, C., Jenkins, O.C.: Robots as web services: reproducible experimentation and application development using rosjs. In: IEEE International Conference on Robotics and Automation (ICRA), pp. 6078–6083. IEEE (2011)
24. Schlett, M.: Trends in embedded-microprocessor design. Computer **31**(8), 44–49 (1998). https://doi.org/10.1109/2.707616
25. Villar, N., Scott, J., Hodges, S., Hammil, K., Miller, C.:NET gadgeteer: a platform for custom devices. In: Kay, J., Lukowicz, P., Tokuda, H., Olivier, P., Krüger, A. (eds.) Pervasive 2012. LNCS, vol. 7319, pp. 216–233. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31205-2_14