Appliancizer: Transforming Web Pages into Electronic Devices

Jorge Garza
Computer Science and Engineering
UC San Diego
jgarzagu@eng.ucsd.edu

Devon J. Merrill
Computer Science and Engineering
UC San Diego
devon@ucsd.edu

Steven Swanson
Computer Science and Engineering
UC San Diego
swanson@eng.ucsd.edu

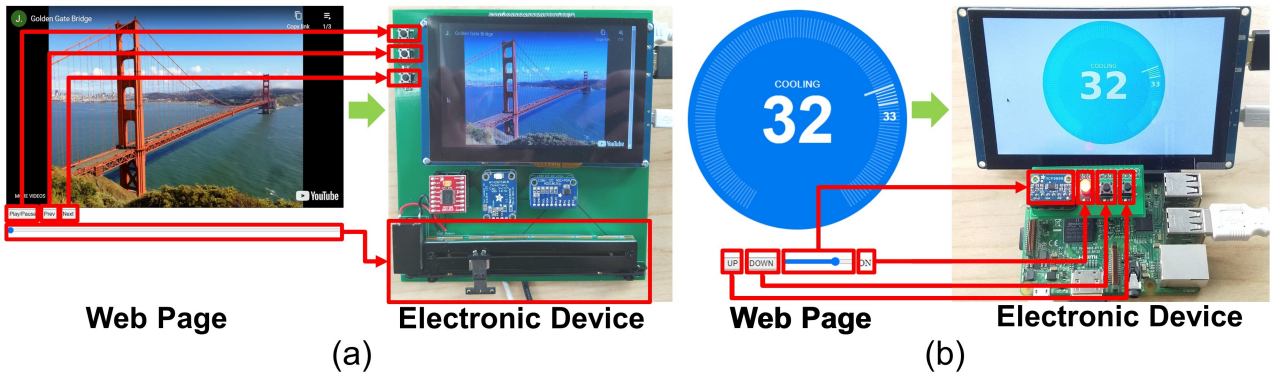

Figure 1: Appliancizer allows web pages to be transformed into fully functional PCB prototypes for rapid prototyping of smart electronic devices. Left a), we show a video player web page that provides a familiar on-screen interface for playing videos. Left b), a thermostat simulated on a web page with basic controls for adjusting the temperature. Right of the web pages shows automatically generated smart electronic devices running the same web application but with a tactile interface.

ABSTRACT

Prototyping electronic devices that meet today's consumer standards is a time-consuming task that requires multi-domain expertise. Consumers expect electronic devices to have visually appealing interfaces with both tactile and screen-based interfaces. Appliancizer, our interactive computational design tool, exploits the similarities between graphical and tangible interfaces, allowing web pages to be rapidly transformed into physical electronic devices. Using a novel technique we call *essential interface mapping*, our tool converts graphical user interface elements (e.g., an HTML button) into tangible interface components (e.g., a physical button) without changing the application source code. Appliancizer automatically generates the PCB and low-level code from web-based prototypes and HTML mock-ups. This makes the prototyping of mixed graphical-tangible interactions as easy as modifying a web page and allows designers to leverage the well-developed ecosystem of web technologies. We demonstrate how our technique simplifies and accelerates prototyping by developing two devices with Appliancizer.

CCS CONCEPTS

• **Human-centered computing** → **Human computer interaction (HCI)**; • **Applied computing** → **Computer-aided design**; • **Hardware** → *PCB design and layout*; • **Computer systems organization** → *Embedded systems*; • **Information systems** → *Web applications.*

KEYWORDS

Web applications, electronic design automation, synthesis, human-computer interaction, HTML, PCB layout

ACM Reference Format:
Jorge Garza, Devon J. Merrill, and Steven Swanson. 2021. Appliancizer: Transforming Web Pages into Electronic Devices. In *CHI Conference on Human Factors in Computing Systems (CHI '21), May 8–13, 2021, Yokohama, Japan.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3411764.3445732

CHI '21, May 8–13, 2021, Yokohama, Japan

ACM ISBN 978-1-4503-8096-6/21/05.
https://doi.org/10.1145/3411764.3445732

1 INTRODUCTION

Electronic devices play an essential role in our modern society. From the automation of many daily tasks, such as thermostats that maintain a comfortable temperature in our home, to multimedia entertainment devices. With the arrival of smart devices, such as smartphones, users expectations of their interaction with electronic devices have changed. Smart devices are expected to have rich interfaces that combine graphical displays with physical buttons and controls. Smart devices must also be connected to the internet and integrated into ecosystems to communicate with other devices [10]. Developing smart devices requires both designing a physical device as well as web-oriented interactions and rich graphical interfaces.

Web technologies such as HTML, CSS, and JavaScript have proven adept at creating interactive interfaces and facilitating integration with online services. The myriad available libraries make

creating compelling animations, multimedia playback, and interacting with web-services possible with a few lines of code. These capabilities make web technologies ideal for designing interfaces for smart devices. For example, Tizen [38] and Firefox OS [31] use web-based architectures for smart devices. However, the many details involved with integrating web technologies with tangible interfaces slows new product development.

The conventional process for developing smart devices that combine graphical and tangible interactions requires substantial expertise in a range of specialties – user interface design, PCB design, and embedded programming [24]. This makes it hard to quickly, agilely, and cheaply prototype these complex devices [27]. Even though the accessibility to manufacturing and assembly services has improved, there is still a gap between having access to manufacturing technology and the skills and knowledge required to build these devices [26].

To bridge the gap, several design tools, such as JITPCB [3], EDG [36] and TAC [1], automate the generation of circuits from software by creating new hardware description languages at the circuit netlist or schematic-level. Enabling the programmatic or visual creation of circuits, without requiring expertise in schematic creation or PCB layout design. However, these tools do not target smart devices (that is, devices with screens and web connectivity). They also require developers to learn novel hardware description languages or syntax. To the best of our knowledge, no tool integrates graphical and physical user interfaces using existing web technologies.

In this paper, we address this shortcoming with Appliancizer, a computational design tool that allows designers to prototype electronic devices as web pages, which are then automatically converted into physical smart devices with tactile interfaces. Appliancizer provides an interactive online IDE[1] that allows users to simulate the interactions between hardware components and on-screen elements. Appliancizer guides users through the device design workflow, abstracting away the complexity of hardware design. With our computational design tool, the process of designing smart devices with both graphical and tactile interfaces (mixed interfaces) is simplified to only GUI and software development.

Compared to previous work such as JITPCB, EDG, and TAC, our approach uses ubiquitous web technologies and its rich GUI Model-View-Controller (MVC) model to create circuits. In Appliancizer, HTML is our hardware description language, therefore there is no need to add any extra configurations or change the source code of the web-based prototypes. This facilitates rapid prototyping of devices with both physical and graphical interfaces. As a consequence, web developers (and others without experience with embedded systems) may more easily create embedded devices.

Appliancizer uses a modular design approach that encapsulates multiple reusable circuit schematic pieces with their required low-level code and other circuit specifications, referred through the paper as *tangible controls* (e.g., physical buttons, sliders, LED's required schematics, hardware interfaces, low-level code, etc). Tangible controls are designed by experts and reusable by novices. Each tangible control maps to one of the three supported HTML GUI elements (buttons, sliders, and text). For example, both knob and temperature sensor tangible controls map to the HTML slider element. By combining the different HTML GUI elements, Appliancizer can automatically generate new circuit board designs and the required low-level code.

[1]Appliancizer interactive online IDE - https://appliancizer.com/.

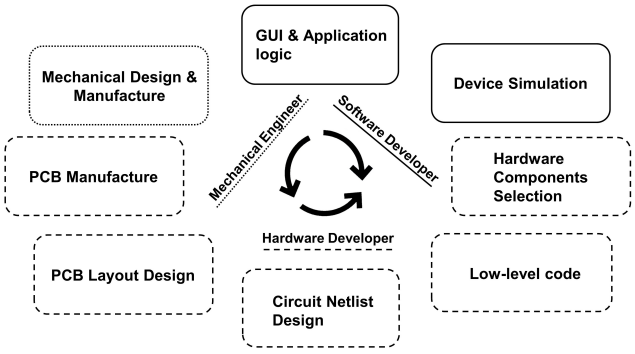

Figure 2: Conventional development process cycle of modern smart devices. Underlined are the tasks required to be accomplished by developers in each field.

Appliancizer uses and demonstrates *essential interface mapping*. A novel technique that allows the automated generation of (i) a simulated interface of the smart device, (ii) schematic and PCB layout ready for manufacture, (iii) the low-level code required to make the device work. Appliancizer also explores the use of a custom web browser for easier application deployment, an approach enabled by our essential interface mapping technique.

Our tool is built on top of our previous work Amalgam [13]. Amalgam is a framework that extends the HTML DOM to the physical domain allowing software communication between graphical and tangible elements. However, Amalgam does nothing to assist the user with hardware design and application deployment. In contrast with Appliancizer, Amalgam users must design the PCB board manually, program the pin mappings, and learn complex Linux commands to run the web application.

The rest of this paper is organized as follows. Section 2 surveys the conventional development process for smart electronic devices with mixed interfaces. Section 3 describes previous work and techniques for accelerating and reducing the complexity of the conventional development process. Section 4 and Section 5 introduce Appliancizer and our essential interface mapping technique. Section 6 demonstrates our tool and technique capabilities by describing two devices made with our tool. Section 7 evaluates with a preliminary user study the rapid prototyping capabilities of devices with Appliancizer. Section 8 discusses the advantages and limitations of our work. Finally, Section 9 presents future work and Section 10 concludes.

2 BACKGROUND

The conventional development process for smart electronic devices with mixed interfaces consists of a variety of tasks that require skills and knowledge in many diverse fields: software, hardware, and mechanics. Figure 2 shows a diagram of the typical development process cycle, as well as the different tasks that we briefly describe below.

GUI & Application logic In this stage, the graphical user interface (GUI) and the software of the device are developed. As embedded device processors are becoming more powerful and priced lower, the use of high-level programming languages (e.g., Python, Java, or web programming languages) is becoming more popular among these devices [14, 19]. Web programming languages (HTML, CSS, and JavaScript) have emerged as a popular solution for designing GUIs on embedded devices due to their capabilities: state-of-the-art for developing rich GUI, first-class networking, and available libraries.

Device Simulation Embedded device software development is tightly coupled with the hardware IO. Device simulation is often necessary to guide the design of the user experience and choose what form the interface should take. Simulation lets designers iterate quickly, start software development before the hardware is ready, and identify potential issues early.

Hardware Components Selection The selection of electrical components is made in this stage. Components such as ICs, LEDs, resistors, capacitors, and sensors are selected based on design specifications and desired device futures. An understanding of the capabilities of each component is required to select each component correctly. User interface decisions drive hardware selection, which, in turn, impacts how the programmer implements aspects of that interface.

Low-level code After hardware components selection, code that makes the connection between the software and each hardware component is needed. We use the term *low-level code* to refer to all the necessary code required to communicate with hardware components.

Embedded computing devices use a variety of wired communication protocols for interfacing with hardware components, such as GPIO, I2C, SPI, and UART. We refer through the paper to these protocols as *hardware interfaces*. Each hardware interface can address a wide range of sensors and actuators, including accelerometers, LEDs, servo motors, and light detection sensors as examples. Embedded computing platforms (e.g., Raspberry Pi [33]) provide groups of pins with different capabilities that may cover some or all of the above interfaces. The pin numbering varies for each embedded computing platform. To make the low-level work with the different computing platforms developers also need to map the low-level code with the pins of the computing platform, known as pin mapping.

The low-level code communicates data between the software application and the hardware components connected to the computing platform. In this paper, we use the term *essential interface* to describe the most essential data transmitted form hardware components to the software application. For example, even if communication with an I2C temperature sensor requires multiple I2C data transfers. The essential interface data that a temperature sensor transmits to the software application is a range of numerical values that indicate the measured temperature.

Circuit Netlist At this stage, connections are created between the different selected hardware components. PCB design tools are used to generate circuit netlists as they allow designers to visually interconnect the hardware components required for a device. The visual representation of circuit netlists is known as PCB schematics. KiCAD [7] and EAGLE [11] are examples of PCB design tools. The pin mappings in the low-level code must match the PCB schematics for the device to function properly.

PCB Layout Design After completing a schematic design, PCB design tools also provide graphical interfaces for creating PCB layouts. A PCB layout digitally describes the physical placement and routing between hardware components. From a PCB layout, manufacturing files can be generated containing all the specifications required for PCB fabrication; known as Gerber files.

PCB Manufacture The cost of manufacturing PCBs has dropped in recent years, allowing hobbyists and students to create high-quality, long-lasting, and reliable circuits. PCB fabrication industries like JLCPCB [20], for example, provide 2-layer PCB fabrication for $ 2 and 1-week lead times, allowing for low-cost rapid iterations of PCB prototypes. After PCB manufacturing, industries can also provide assembly services for the physical placement and soldering of hardware components, freeing hardware developers from the technical skills required to assemble these devices.

Mechanical Design & Manufacture At this stage, the manufacture of enclosures and mechanical parts is carried out. Mechanical engineers must know the dimensions of PCBs and hardware components to design mechanical parts. Connector openings, gear sizes, and screw terminal positions can go wrong if they are not known. While the mechanical parts of the electronic device are essential, in this paper we only focus on the software and hardware tasks of the design process.

The conventional development process requires many steps and deep experience in different areas adding extra costs. Because the process is highly coupled, any design change must be reflected at all stages of development, making iterations time-consuming and complex, especially as each task is typically handled by different people with different skill sets.

3 RELATED WORK

In this section, we review previous work that helps reduce the complexity of the design process required for the development and prototyping of electronics devices, as well as techniques relevant to this paper.

3.1 Rapid prototyping of electronic devices

Developing electronic devices is a complex process, therefore it is important to place our design tool in the context of the field. In this section, we discuss existing systems and how they interact with the different stages of electronic device development.

Several non-PCB prototyping systems have developed solutions to accelerate one or more steps of the conventional development process of electronic devices described in Section 2.

3.1.1 Tangible interface simulation. Many commercial tools [29, 34] offer low-level circuit simulation, targeted toward expert users. ThinkerCad Circuits [42], in particular, provides an accessible graphical environment allowing less-experienced users to simulate hardware component behavior and low-level code using a virtual breadboard.

Compared to these tools, Appliancizer uses higher-level abstraction. Appliancizer simulates interface behavior, not electrical circuits. Only essential interface events between application software and the hardware are simulated.

3.1.2 Rapid circuit prototyping using modular design. In modular design, hardware and software complexity is divided into different modules that can be reused. Modular design techniques are commonly used to achieve faster prototyping compared to the conventional development process. Multiple authoring tools allow rapid interactive exploration of circuit designs at the physical level using a modular design.

Microsoft .NET Gadgeteer, MakerWear, littleBits, and SoftMod [6, 22, 23, 44] use prefabricated circuit modules as electronic building blocks that can snap together to create a range of electronic devices. These systems rely on proprietary toolkits. This limits support to only the proprietary hardware modules created explicitly for these systems. While toolkits attempt to standardize modules (for example, Grove [16]), these proprietary systems may be subject to limited future support and licensing fees.

In contrast, Appliancizer creates custom PCB prototypes using the universally used PCB schematics as modules. These PCB schematics can be directly used or modified to create a final commercial device. Additionally, schematic modules can easily be added to our tool as EAGLE files and updated without requiring expert designers to go through the additional steps to manufacture them. Appliancizer schematic modules also use the same off-the-shelf components (e.g., buttons, sensors, LED's, etc) that will be used in the final device.

3.1.3 Accelerating low-level code development. Modular design is also used for low-level code in .NET Gadgeteer, Arduino, and others [2, 5, 17] by providing higher-level libraries. These systems successfully remove some of the expert knowledge needed to program embedded devices (e.g. microcontroller registers manipulation) but still require user implementation of low-level communication with tangible interface components.

Appliancizer is built on top of Amalgam [13]. Amalgam deeply integrates different low-level code modules into the HTML and CSS syntax creating a clear separation between software code and low-level code. With Amalgam, the low-level code modules can be mapped to trigger HTML DOM events (e.g. onclick), therefore requiring zero low-level code development. However, Amalgam by itself still requires the user to know other low-level details such as pin mappings and Linux device file names.

Amalgam allows Appliancizer to present the same JavaScript events for tangible interface components as graphical interface components. However, by using tangible controls and essential interface mapping Appliancizer completely abstracts all low-level details, making developing tangible interfaces as easy as developing graphical interfaces in HTML and JavaScript.

3.1.4 Mechanical design and mixed interfaces. Makers' Marks [37], RetroFab [35], and .NET Gadgeteer SolidWorks pluginprovide rapid design facilities for the integration of mechanical parts (and device enclosures) with tangible interface components. These systems often make a clear division between the mechanical, hardware, and software domains resulting in a mental gap for designers.

f3.js [21] works across domains and provides designers with a single codebase allowing users to interactively design the device enclosures with the aid of an IDE in conjunction with the device software code. This technique provides a mixed interface generation from a single codebase where one part of the code is utilized for the device functionality and the other for the device mechanical design.

Compared to these tools, Appliancizer focuses on helping users with the PCB design of the device and not on the mechanical design. Similar to f3.js, our tool provides a single codebase for the generation of a mixed interface interaction with the electronic device.

Amalgam allows Appliancizer to present the same JavaScript events for tangible interface components as graphical interface components. However, by using tangible controls and essential interface mapping Appliancizer completely abstracts all low-level details, making developing tangible interfaces as easy as developing graphical interfaces in HTML and JavaScript.

3.2 Automating PCB development

Current PCB design tools operate at lower abstraction levels, limiting the design process to the circuit netlist and PCB layout design tasks described in Section 2. Furthermore, as mentioned by Lin et al. these tools have stayed fundamentally the same over the years, without providing design assistance or improvements for novices [24]. Recent work has proposed integrating the high-level abstractions of circuit design specifications into programming languages to allow the automation of multiple design steps. This transformation process is similar to the synthesis of hardware description languages (e.g., Verilog [41] and VHDL [30]) into gate-level blocks (for example, NAND, NOR, Flip-Flops) but for board-level circuits. Hence, this process is also called synthesis for circuits. Synthesis for circuits translates high-level hardware description languages into circuit netlist or schematic modules of commonly used circuit designs (for example, an LED with a resistor) instead of gate blocks. The benefits of synthesis for circuits include: accelerated development; avoiding the need for in-depth knowledge of electronics; and opening electronic design to new communities with a different set of skills, such as software developers.

Just in Time Printed Circuit Board (JITPCB) [3], Embedded Design Generation (EDG) [36] and Trigger-Action-Circuits (TAC) [1] are examples of tools that go beyond the traditional schematic capture process and provide board-level hardware description languages that allow synthesis for circuits. However, despite achieving faster development times and a notable reduction of complexity in comparison to the conventional development cycle, these synthesis tools integrate the hardware abstractions at the software level, requiring developers to learn extra development steps or limiting the functionality of the software application. JITPCB, for example, requires users to learn a separate programming language, Stanza [32], and manually specify which schematic modules to use

in conjunction with the software. EDG requires users to learn an API to generate circuit diagrams. These two tools have benefits to experts doing low-level circuit design. TAC further abstracts the circuit specifications as a visual programming language but limits the application logic complexity by only having an if-then programming model [43]. In contrast to these systems, Appliancizer uses a well-developed programming language specifically designed for creating UIs with a focus on network interactions (HTML and JavaScript).

Appliancizer integrates the hardware abstractions at the HTML GUI level and leverages its rich MVC model to create circuits. In Appliancizer, HTML is the hardware description language, therefore, our tool is also defined as a GUI-based hardware description language for board-level circuits. Combining this abstraction with our essential interface mapping allows for circuit synthesis without requiring users to learn extra development steps or even modify the software application source code. This allows Appliancizer users not to have to design schematics or PCBs layouts to create responsive, tangible interfaces

At the level of workflow automation, and taking as a model the diagram described in Figure 2, of the synthesis tools described above, only JITPCB can generate full PCB layout designs from a software programming language. EDG and TAC limit their work at the circuit netlist design step. Appliancizer's essential interface mapping technique allows workflow automation that spans from the device simulation to the PCB layout design tasks of the smart devices development process. Our tool generates complete PCB Gerber files that can be sent to manufacture without modifications.

3.3 Mapping graphical and tangible interfaces

A modular design concept that enables rapid prototyping has prevailed for the computational design of tangible interfaces that span from desktop computers to embedded devices. Especially post-WIMP, many of these tools have exploited the correlations that exist between graphical and tangible interfaces to accelerate prototyping.

Phidgets [15] first proposed packaging hardware components and their required low-level code into modules that can interact with on-screen GUI elements, allowing users to prototype tangible interfaces faster. Similarly, d.tools [18] uses the concept of phidgets to graphically prototype hardware and screen transitions of electronic devices. D.tools exploited these similarities by creating physical and virtual duals of components that are mapped to make tangible components interact with graphical ones. Further exploration of the mapping between graphical and tangible interfaces is done by iStuff [4] and the work by Coyotte et al [9] to accelerate the prototyping of tangible or mixed interfaces. These prior tools rely on a full desktop event handling system for mediating between physical device components and the software application logic. Appliancizer allows users to first prototype smart devices as GUIs and then generate a fully functional physical version with minimal effort. Appliancizer is, to the best of our knowledge, the first system to use graphical-tangible interface mapping to fully automate the hardware design of standalone smart devices.

4 APPLIANCIZER

To lower the PCB design entry bar and speed up the prototyping of mixed graphical-tangible interactions required for smart devices design, we created Appliancizer. Appliancizer is an online computational design tool that enables designers to transform web pages into fully functional PCB prototypes of smart electronic devices. With our tool, HTML-based interfaces are transformed into mixed interfaces, interfaces that include GUI and tactile elements. Compared to the conventional development process that requires designers to have extensive expertise in multiple fields, Appliancizer simplifies smart device PCB prototype development by requiring only front-end web design knowledge and minimal hardware knowledge. In addition, we release Appliancizer's implementation as open-source software[2]. In this section, we provide an overview of the design process with our tool.

4.1 Design process overview with Appliancizer

Appliancizer simplifies the design workflow required to prototype smart device PCBs by providing simple steps to build these devices. Figure 3 shows the interface of the tool. Designers first start by developing a front-end web application consisting of HTML, CSS, and JavaScript code, much as they would for a normal web-based application. Our tool provides a built-in front-end web editor for user convenience which can be accessed with the add a new web page button (a). Designers create their web-based prototypes with the clear idea that the prototype will be converted into a device and use the three supported HTML elements as mock-up hardware components to begin prototyping the interactions between graphical and potential tangible interfaces. The three supported HTML elements, listed in Table 1, are defined as *HTML controls* through the rest of the paper.

After the new web-based prototype is developed, users can then upload the web application to a virtual screen display within our IDE that represents what users will see on the smart device screen. Our tool will highlight a green dotted border around the HTML controls that can be transformed into tangible interfaces. Otherwise, a red dotted border will be displayed (b).

The available HTML controls to transform can be dragged and dropped into a resizable virtual PCB (c) and (d), creating a clear separation between the graphical interfaces and the potential tangible interfaces. Each HTML control dropped into the virtual PCB maps to tangible controls which the user can select (e). Appliancizer supported tangible controls are listed in Table 2. Examples of tangible controls that the user can select are tactile buttons, LED's, and temperature sensors hardware components.

Accompanying components like resistors or any other hardware components required by the main tangible control hardware component are also automatically added to the virtual PCB. A simulation mode enables developers to simulate the potential tangible interfaces of the smart device (f). Finally, with a simple click, developers can get the list of all parts with a link showing where to buy them, Gerber files ready for manufacture automatically generated from the virtual PCB, and simple steps to deploy and run the web application on the new smart device d). The generated smart device for the virtual PCB in d) is illustrated in Figure 1 a).

[2] Appliancizer source code - https://github.com/NVSL/Appliancizer.

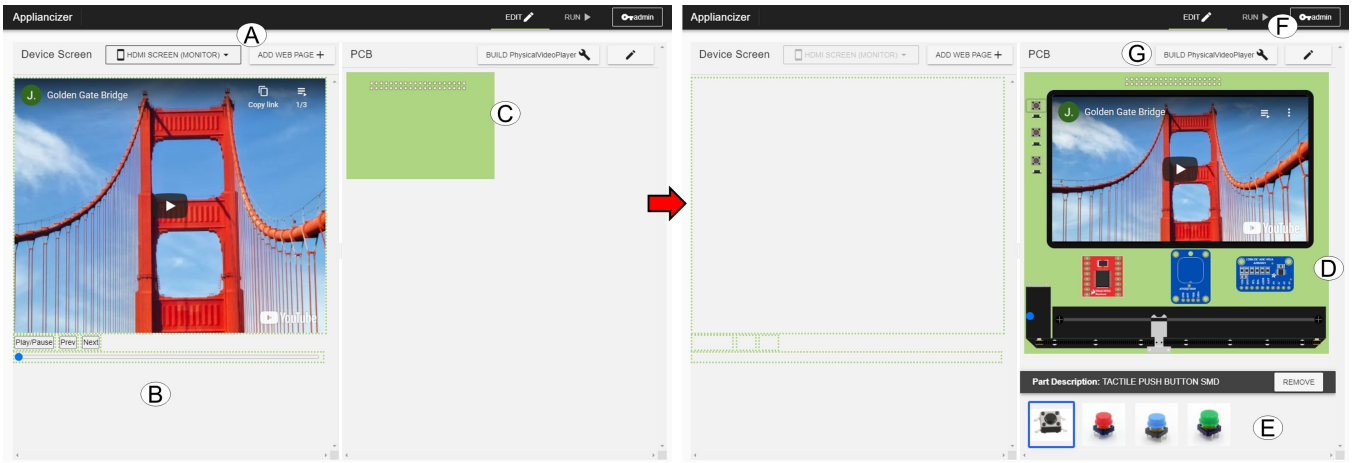

Figure 3: Appliancizer IDE. Figure shows a) button for adding web applications, b) virtual screen display area, c) empty virtual PCB, d) virtual PCB with virtual hardware components, e) tangible controls hardware components selection area, tangible interface simulation f), and a build button for generating the PCB layout, display required parts, and show application deployment steps g).

4.2 Graphical to tangible interface process overview

With Appliancizer, HTML controls can be moved off-screen to become tangible interface components. Figure 4 shows a more detailed view of the process for transforming a graphical interface into a tangible one by showing how internally a button and span HTML controls are transformed into tangible interfaces with which a user can interact. After the HTML controls in a web page a), b) are dragged and dropped into the virtual PCB c). Our essential interface mapping matches the essential IO data similarities that exist between dragged HTML and similar hardware components. An HTML control button, for example, matches tactile pushbuttons perfectly as both produce an ON or OFF essential input data.

For each hardware component and accompanying components in the tangible control module, schematic modules describing the interconnection between the components are provided by our tool. Each schematic module represents an HTML control. By combining the different HTML controls, new circuit board designs can be generated. When ready for manufacturing, a complete schematic is generated from each separate schematic module d). A PCB layout is then generated from the complete schematic e). Finally, f) shows the final PCB after manufacture.

Using a modular design approach our tool also encapsulates the required low-level code for communicating with each hardware component. For deployment, a web application is also generated with the required low-level hardware code allowing its interaction with hardware components. A custom operating system image, provided by our tool, allows users to navigate and run the web application. Finally, at runtime, the selected HTML elements are replaced by *hard HTML elements* [13] consisting of web components [45] and low-level code that interface with the selected tangible controls hardware components. Figure 1 shows two examples of smart devices, a video player and a thermostat. Its PCBs were automatically

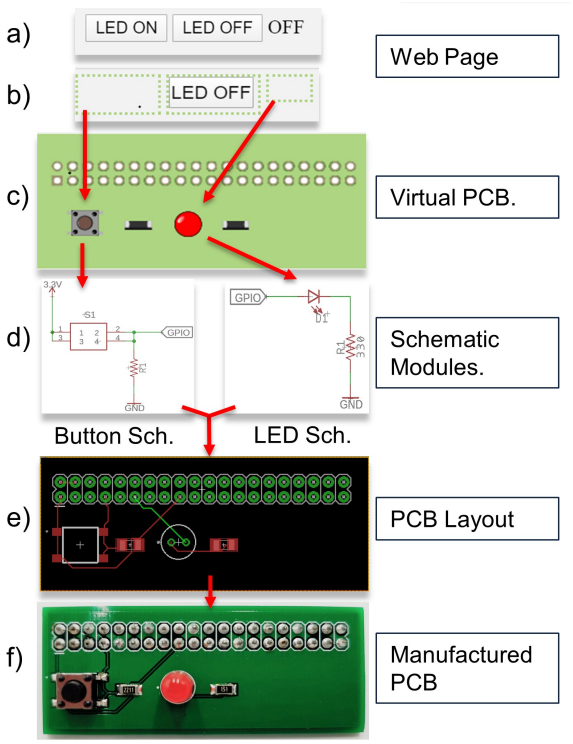

Figure 4: Graphical to tangible interface process. Two HTML controls a `<button>` and `<span>` are transformed into tangible interfaces while one HTML element, a `<button>`, is left as a graphical on-screen interface element.

created from a web page, and their hardened HTML elements allow physical interaction with the device's web application.

4.3 Appliancizer tangible controls

To better explain our tool, we define as tangible controls as the encapsulation of high-level circuit specifications for hardware components (e.g. LEDs, pushbuttons, etc). The circuit specifications that are encapsulated into each tangible control and the method of integration with development tools affect the system capabilities. Examples of circuit specifications that can be encapsulated are low-level code for interfacing with the hardware components, circuit netlists, voltages, and hardware communication protocols required by the hardware parts, among others.

Figure 5 on the right shows an overview of all the circuit specifications that are encapsulated for each tangible interface component provided by our tool. Each Appliancizer tangible control consists of a schematic module, a low-level code block, required hardware components, and the hardware interfaces required to communicate with the hardware components.

The tangible controls concept has been implemented in previous tools, described in Section 3.2 and Section 3.3, each encapsulating different hardware features. Phidgets and Amalgam [13] for example have explored the use of tangible controls, but limit the encapsulation to contain only low-level code, leaving the PCB schematic and layout design to the developer. In contrast, Appliancizer tangible controls provide additional circuit specifications that allow the generation of PCB schematics and layouts. JITPCB, EDG, and TAC which similarly allow synthesis, limit the integration and exploration of tangible controls to the software level. On the other hand, Appliancizer integrates tangible controls at the graphical interface level, allowing the essential interface mapping of graphical and tangible modules.

Appliancizer tangible controls are designed by experts and reusable by novices. Compared to existing non-PCB rapid prototyping tools, experts designing these modules can add and update the modules as familiar plain-text PCB schematic files, rather than designing and manufacturing toolkit-compatible modules. Appliancizer supported tangible controls are listed in Table 2.

5 ESSENTIAL INTERFACE MAPPING

By forming well-established mappings the translation between graphical and tangible elements, or to be more precise between HTML controls and tangible controls, can be performed, enabling design automation of smart devices that require rich GUI. In previous works, iStuff [4] and the work of Coyotte et al [9] limited the exploration of interface mappings at the user interface level. Therefore, preventing the analysis of hardware components that do not require direct physical interaction with the user (e.g, a temperature sensor).

While, from the perspective of HCI is not necessary, for fabrication the goal is to map as many HTML elements with hardware components as possible, even if the user never interacts physically with the hardware components. A mapping that covers a wide range of hardware components allows the creation of more diverse smart devices. Therefore, to cover a wider range of hardware components we suggest mapping HTML controls and tangible controls at the essential interface level. Opening the analysis of how graphical interface elements can also be used to prototype the behavior of hardware components that do require direct physical interaction.

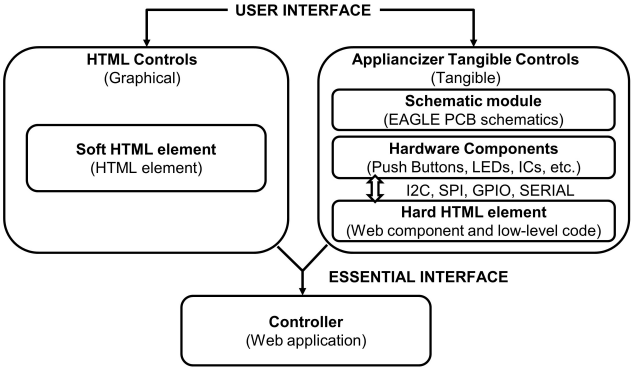

Figure 5: HTML controls vs Appliancizer tangible controls: To consider the wide range of hardware components we considered mapping at the essential interface level, thus encompassing hardware components that may not interact with the user, such as sensors.

Table 1: Mapping between HTML controls and hardware components

HTML controls	*Essential interface*	*Hardware components*
`<span>`	Multi-byte Output	**Actuators**, LEDs, Relays, Servos, LCD Displays, etc.
`<input type="range">`	Numeric Input & Output	**Sensors**, Temperature Sensor, Potentiometers, etc.
`<button>`	1-bit Input	**Interrupts**, Push Buttons, Status Signals, etc.

Figure 5 shows a comparison between HTML controls and our Appliancizer tangible controls using the Model-View-Controller architecture model as a reference. Both the HTML controls and the communication of the Appliancizer tangible control with the controller are abstracted at the level of the essential interface, or the most essential data that is sent to the web application.

By studying the similarities of the HTML elements and essential interfaces of the different hardware components, we created a relational table that contains a mapping between the two. Table 1 shows our mapping table that we organize between HTML controls and a classification of different hardware components (e.g., sensors, actuators, etc.). One of the things we considered when creating this mapping is being able to address as many hardware components as possible with the least amount of HTML elements. This makes application development easier by not requiring specialized HTML elements to match specific hardware components. We describe each essential interface mapping in detail below.

5.0.1 Multi-byte Output. A `<span>` or text HTML element provides information to the user. Therefore, we consider this element to be mapped to actuators that require one or multiple bytes of information sent by the web application to react, such as LEDs, servos, and LCDs. Once the interface becomes tangible, the data

is transmitted to the low-level code and transformed accordingly for each hardware component. For example, for a LED, the web application must produce an "ON" or "OFF" text to turn the LED on or off. In the case of a servo motor, the number of text in angles (e.g. "45") will make the servo rotate 45 degrees. Similarly, a 16×2 LCD screen will only display the raw text produced by the software application.

5.0.2 Numeric Input & Output. `<input type="range">` range sliders naturally match any sensor that produces a range of values. These sensors can vary from physical knobs or sliders that require human interaction to effect changes to temperature, light, and humidity sensors that interact with the environment. While range sliders are perfect for providing input data, web applications can also change the slider position through code, making the essential interface also a good candidate for outputting data to hardware components that require a range of values to actuate (e.g. servos).

5.0.3 1-bit Input. The `<button>` HTML element remains as the most intuitive for providing 1-bit input data to software applications. However, hardware components that produce 1-bit inputs are not limited to tactile pushbuttons. Integrated circuits that trigger interrupts when special conditions are met can be used as well.

5.0.4 The screen component. Mixed interface smart electronic devices require the use of a display to allow digital interaction with the device. Devices made with Appliancizer by default will display all the HTML GUI elements left in the virtual screen display. Developers can then connect any HDMI display to the device to interact with HTML elements on the device screen. However, in certain applications, the integration of the device screen with the PCB is required. For these applications, Appliancizer allows the `<iframe>` element to be dropped inside the virtual PCB where different HDMI displays with varying sizes can be selected allowing developers to populate the virtual hardware components adjusting to the screen display dimensions, as is illustrated in Figure 3 d).

5.1 Appliancizer tangible controls library

The implemented Appliancizer tangible controls that can be selected for each HTML control, listed by their essential interface, is shown in Table 2. Each off-the-shelf hardware component that can be used with our tool can come in different sizes and with variations, as is the case with LEDs that can come in thru-hole or surface mount device (SMD) versions and varying colors. With Appliancizer, low-level code can be written once and reused for similar versions of the hardware part. However, a schematic module must be created for each part to address the changes in sizes and shapes.

While some of our tangible controls use solderable breakout boards to facilitate assembly. With our approach, each discrete electronic component soldered onto the breakout board PCB can be directly integrated into the Appliancizer-generated PCB layout, as we do with the button and LED tangible controls.

5.2 Simulation of the device tangible interface

Device simulation enables developers to prototype and test the software of the device even before the device is manufactured, allowing experimentation and avoiding repeated prototyping to analyze the system behavior. Full simulation of the device hardware is expensive in terms of the computational resources needed. A popular and less expensive technique is to display an on-screen mock-up of the tangible interfaces that allow software developers to trigger simulated essential interface data events sent to the software application by the hardware components. This approach is common in mobile application development [25], where virtual buttons and sensors are used to represent physical components such as volume, home, keyboard buttons, and sensors allowing developers to debug their mobile applications without requiring a physical device.

Table 2: Appliancizer tangible controls library

Tangible controls	*Essential interface*
LED (Red, Green, Blue) Thruhole & SMD	Multi-byte Output
Servo Motor	Multi-bytes Output
16×2 LCD Display	Multi-bytes Output
Potentiometer	Numeric Input & Output
Motorized Potentiometer	Numeric Input & Output
Temperature Sensor	Numeric Input & Output
Tactile Push Button Thruhole & SMD	1-bit Input

Our essential interface mapping technique inherently allows the creation of a virtual tangible interface for smart devices, since both, HTML controls and Appliancizer tangible control, are matched by the same essential interface data. After the HTML controls are removed from the virtual screen display and dropped into the virtual PCB area, developers can interact with the virtual hardware components placed in the virtual PCB when in simulation mode. Changes to the virtual hardware components will produce the same effects in the web application since it's the same web GUI but split into a graphical and a simulated tangible interface by the developer. Taking as an example the virtual button in Figure 4 (c), in simulation, any 'onclick' events will still be sent to the controlling web application code. Then, after PCB fabrication (f) and deployment of the application, pressing the physical button will trigger the same events in the web application as its virtual counterpart.

5.3 Automated PCB layout generation

Adding circuit characteristics to tangible control modules also allows Appliancizer to automatically generate the PCB layout required for the smart device. The PCB layout generation is divided into three tasks: hardware component selection, complete schematic generation, and hardware components placement. Below we describe these tasks and conclude with the PCB manufacture and assembly process.

5.3.1 Hardware component selection. Once the HTML controls are placed on the virtual PCB, developers can select among the hardware components implemented in Table 2, only those that match the essential interface mapping. Figure 3 (e) shows an example of the hardware components that can be selected for the `<button>` HTML control.

5.3.2 Complete schematic generation. To automatically generate the complete schematic we use a modular design technique at the

circuit level. In this technique, schematic pieces of commonly used electronic designs (e.g., LED and resistor, etc) are separated into modules, as is shown in Figure 4 (d). Each schematic module consists of a circuit with power nets, ground nets, internally connected nets, and disconnected nets.

Internally connected nets describe the connection of the main tangible interface components (e.g, a tactile button) with their accompanying hardware components (e.g, resistors, capacitors). Disconnected nets are left for the hardware interfaces (e.g., GPIO, I2C, etc) required for each hardware component. At build, each disconnect net is greedily assigned to the next available hardware interface of the computing device platform (e.g., GPIO interface to pin 4). In case there are no more hardware interfaces can be assigned we alert the user. Similar circuit netlist or schematic generation techniques have been implemented by JITPCB, EDG, TAC, and more recently Ehidna [28] for devices that use microcontrollers. In our tool, however, we target systems on chips (SoC) computing devices that can run operative systems which have become more popular in new embedded designs [12].

5.3.3 Hardware components placement. To generate a PCB layout the physical position of hardware components and the size of the PCB must be known. After the HTML controls are dropped into the virtual PCB and the desired hardware component is selected, a 2D virtual part image that represents the physical dimensions of the real hardware part can be positioned by the designer in the resizable virtual PCB. Accompanying hardware components for each Appliancizer tangible control can be moved as well. After virtual placement, the position of each virtual hardware component and the size of the virtual PCB are then used to create the PCB layout, described in more detail in Section 5.3.4.

Compared to tools like JITPCB, which require adding placement and rotation of hardware components programmatically, with Appliancizer, designers visually move and place the virtual hardware parts in their desired locations inside the virtual PCB. Although the hardware components placement could also be generated automatically, we choose to use a WYSIWYG methodology as we believe is more intuitive for the user.

5.3.4 Generating the PCB layout. After the complete schematic is generated and knowing the hardware components positions, the PCB layout generation process takes place. From the complete schematic, we generate the PCB layout positioning all hardware components inside the PCB to match the virtual PCB design. After placement, we run the auto-routing process provided by EAGLE, a third party PCB CAD used by our tool, to generate the routed PCB. Finally, after the PCB layout routing completes, the Gerber files are also generated using EAGLE. The Gerber files are then sent to a PCB fabrication service for its manufacture. Figure 4 shows in (e) the PCB layout after is routed, and (f) the PCB after manufactured. The physical hardware parts were assembled manually, however, PCB services can also provide assembly. To merge the different schematic modules and generate the PCB layout we use Swoop [39], a tool we created to edit, create, and manipulate EAGLE files. Only the automatic routing process and the Gerber file generation tasks are performed using EAGLE.

5.4 Automated low-level code generation

At build time, our tool also generates all the low-level code required to make the smart device work. The generation of the low-level is divided into two steps: A pin mapping process and the generation of the user web-application integrated with the required low-level code for the device.

5.4.1 Pin mappings. Appliancizer also generates hardware interfaces mappings at the software level, known as pin mappings. The pin mappings indicate which IO pins each tangible control low-level code should use and must match the schematic of the device (e.g., button GPIO interface to pin 4). tangible control are not limited to one hardware interface, Figure 1 device (a) shows a motorized potentiometer which requires two GPIO interfaces to move the motor and one I2C interface to get its position.

To automatically generate the pin mappings each tangible control is encoded with the hardware interfaces required by the low-level code to communicate with the hardware components. In the same complete schematic generation process described Section 5.3.2, the required hardware interfaces by each tangible control are greedily assigned to the next available computing device header pins. However, in this case, the resulting product is a Amalgam-enhanced CSS file used by Amalagam to map the low-level of each tangible control with the correct hardware pins and that match the generated schematic. The generated CSS file is used by Amalgam for the *hardening* [13] process to take place.

5.4.2 Web application low-level code integration. After the pin mapping process, we add to the user web application all the low-level code from each selected tangible control. Finally, the user web application, integrated with the required tangible control modules and a CSS file indicating to Amalgam how to harden the soft HTML elements (HTML controls) with the correct tangible control, is hosted on our server. A web link to the hosted web application is provided to the user which they can use to navigate to the generated web application integrated with low-level code.

5.5 Application deployment

The conventional application deployment process requires users to learn complex Linux shell commands for application deployment, making it challenging for novices. To facilitate the process, we created a custom OS that allows users to simply navigate to the generated web link described in Section 5.4.2. Links are shareable making it possible for other users to test applications created by others. Our custom OS can be downloaded from our tool.

At boot, our custom OS provides a familiar web browser interface where users can introduce the generated web link to navigate to the generated web application. Our custom OS also contains Amalgam which is executed when a new web page is loaded, making our web browser behave differently. Unlike standard web browsers, our custom OS web browser will harden the selected HTML control by the developer, extending the HTML DOM to the physical world and allowing physical interaction with the web page. The hardening process is described in depth in [13].

5.5.1 A smart device or a web page? One of the unique characteristics that our essential interface mapping technique offers is the capability of having two versions of the same application, one fully

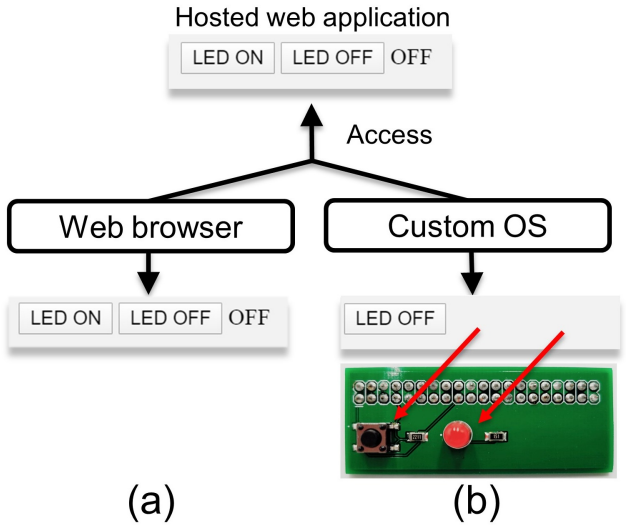

Figure 6: A essential interface mapping technique allows the same application to be accessed as a fully graphical interface application (a) or a mix of graphical and a tangible interface (b).

graphical and the other one with a mix of graphical and tangible interfaces. After our tool generates the web application integrated with low-level code, the web application is still accessible from a standard web browser and will render all HTML elements on the screen. However, when the same web page is accessed through our custom OS the user-selected HTML elements to transform are removed from the screen and their interface becomes tangible. Figure 6 shows the same web application described in Figure 4. Accessing the web application from a standard web browser will result in a fully graphical interface (a). Otherwise, if the web application is accessed from our custom OS with the generated PCB mounted, will result in a fully functional smart device (b).

6 EXAMPLES

To demonstrate our essential interface mapping technique we created two smart devices, a physical video player and a smart thermostat, using Appliancizer. Figure 1 illustrates these two electronic devices. Below we describe each smart device in detail.

6.1 Physical Video Player

The first smart device we created is a physical video player which was transformed into a smart device directly from a video player web application. To create the smart device we first built a soft video player using the YouTube API [46] by following a tutorial available online. Then, we loaded the application to Appliancizer and moved the player HTML controls to the virtual PCB, as shown in Figure 3. Finally, we sent the generated PCB to manufacture resulting in the smart device shown in Figure 1 (a). The physical video player device mimics all the behavior of the original soft version, including the progress bar which moves as the video progresses and which the user can actuate to change the video to a specific time. The user can press the physical push buttons to play or stop the video, and move to the next or previous video. No modifications were made to the source code of the web application for deployment.

Originally our physical video player only had a play and pause button. After testing the video player smart device we wanted to include physical previous and next buttons. Since our essential interface mapping technique makes web applications work as web-based prototypes of smart devices. The buttons were simply added as developing any other web page, without requiring adding low-level code or extra configurations. The old device PCB was replaced by the newly generated PCB and the custom OS web browser was simply refreshed to update the mappings, allowing physical interaction with the added buttons.

6.2 Smart Thermostat

For the second smart device, we took inspiration from the Nest thermostat [40] that provides an intuitive GUI for controlling the temperature inside a room. However, instead of following a tutorial and writing the code ourselves, we decided to search existing web applications from CodePen [8], a popular online editor for sharing front-end web applications. We found that the use of HTML elements as hardware mock-up components is an approach intuitively used for prototyping smart devices graphically and quickly discovering a web-based NEST prototype [8]. Although, the initial proposal of the web application shared by the community was to graphically simulate the Nest thermostat tangible interface, with Appliancizer we transformed the web application into a real smart thermostat.

The soft version of the smart thermostat includes two HTML buttons to increase or decrease the target temperature, an HTML span tag that indicates if the weather to turn ON or OFF the cooling compressor, and finally, a range slider that simulates the ambient temperature of the room. Correspondingly, the built physical version of the devices consists of two physical buttons, a LED that physically simulates the ON and OFF state of the compressor, and a temperature sensor that indicates the ambient temperature with a range of values. However, compared to the motorized potentiometer that also provides an interface to programmatically control the position of the slider using a DC motor, the temperature sensor is limited to only transmitting an input range of values to the web application. Therefore, any initial value set to the soft slider version will be ignored by the low-level code of the temperature sensor tangible control. Finally, for this device we left the screen component as a soft element, allowing the PCB to be placed at the back or at a different angle from the screen display.

7 PRELIMINARY STUDY

To gauge how our essential interface mapping technique speeds prototyping of smart devices that require mixed interfaces we ask two developers with prior experience designing PCBs to build the same smart thermostat described in Section 6.2 using both Appliancizer and Amalgam. The web application of the smart thermostat was provided for both tools. For Amalgam, the hardware footprints library was provided as well, requiring users to only select the hardware components for laying out the PCB.

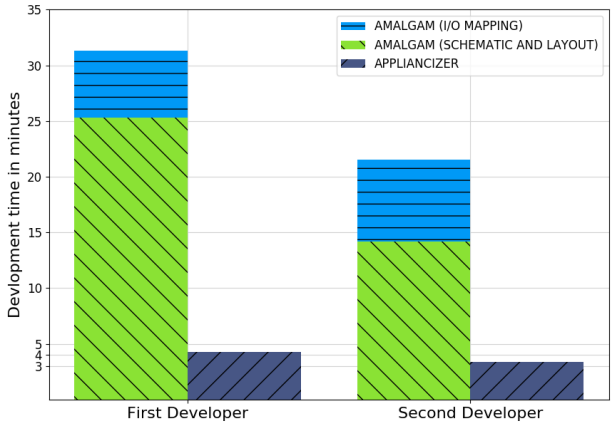

Figure 7: Development time comparison between Appliancizer and Amalgam for designing a smart thermostat.

Figure 7 shows the development time difference between Appliancizer and Amalgam. It took developers an average time of 4 minutes with Appliancizer versus an average of 26 minutes with Amalgam to design a smart thermostat device. A speed-up of 6.5X in development time with Appliancizer. These results were expected since Amalgam, compared to Appliancizer, requires users to generate the schematic and PCB layout manually. Furthermore, with Amalgam, users also require to program the pin mappings for the low-level code to work with the selected computing device. These two separate tasks are shown correspondingly for Amalgam in Figure 7.

8 DISCUSSION

The conventional design, test, and prototype process for smart devices that require mixed graphical and tangible interfaces are composed of complex tasks. To mitigate the challenges we introduce a essential interface mapping technique that allows transforming HTML GUI elements within web pages into tangible ones. To demonstrate our technique we created Appliancizer, an online computational design tool that allows designers to generate functional smart devices from web-based prototypes. In this section, we discuss some observed capabilities and limitations of our essential interface mapping technique and compare Appliancizer to previous work.

8.1 Rapid prototyping of smart devices

While previous tools have achieved synthesis for board-level circuits [1, 3, 36], they limit the high-level hardware abstractions of circuits at the software code level, requiring developers to learn complex integration steps or limiting the software application functionality. In our work, we consider the abstraction at the graphical interface level for smart electronic devices that require graphical and tangible interfaces. Previous tools at this level have demonstrated a relationship between graphical and tangible interfaces [4, 9, 15]. However, they are limited to systems that require the desktop computer as a mediator and are incapable of generating full stand-alone devices. In contrast, our tool takes advantage of these similarities for synthesis, allowing us to transform graphical interfaces into tangible ones and automate different tasks of the design process, without requiring developers to change the application source.

Using a essential interface mapping technique, the development of a smart device comes down to selecting which HTML elements to transform and selecting the desired hardware components that match the essential interface of the selected HTML elements, enabling rapid prototyping down to the circuit board level design.

Section 7 demonstrates Appliancizer rapid prototyping of smart devices. Amalgam developers spend a significant part of their time creating schematics and PCB layouts of the same circuit blocks. Comments from one of our users acknowledge that

> "even remembering how to connect commonly used circuit designs takes time"

After comparing the PCB layouts generated automatically by Appliancizer and the ones manually designed by both users, we discover no significant differences in the routing or design.

8.2 Fast iterations

Compared to the conventional design process of standard PCB software tools that require developers to re-design the PCB and implement new low-level code after adding or removing hardware components, our tool, on the other hand, provides a faster iteration process by automating these tasks. With Appliancizer users only need to choose to leave some items inside the web page as graphical elements and make others tangible, enabling fast iterations of the same smart device design.

8.3 Web-based prototypes

Aside from board-level rapid prototyping, our essential interface mapping technique enables users with the capability to rapidly explore physical prototypes using HTML GUIs, also referred to as web-based prototypes. For our smart thermostat application, even if the web community of original developers for the smart thermostat web application ruled out the idea of having a physical version of the device, they were able to efficiently explore the physical interaction with the device using HTML elements as mock-up hardware components. Even before the existence of our tool, the use of GUI elements as mock-up hardware components seems intuitive for designers for prototyping smart device's physical interactions with on-screen elements. Appliancizer's simulated tangible interface provides the user with a clear division of on-screen HTML elements and those used as mock-up hardware components. This allows a clearer exploration of the interactions between the graphical and tangible interfaces of the smart device. After interaction design exploration using the simulated tangible interface, our tool can then transform the mock-up hardware components into real tangible interfaces.

8.4 Web browsing smart devices

Another observed ability of our essential interface mapping technique is the ability to have two versions for the same web application, one that interacts fully digitally with the user and another in which some HTML elements can physically interact. Since both versions are encapsulated in the same application, applications created with Appliancizer inherit all the properties of web technologies.

Therefore, allowing smart device applications to also be requested, displayed, and executed using standard web browser techniques. Our custom operating system implements the idea by allowing users to navigate to web applications that contain the low-level that makes the smart device work. To run the physical video player and smart thermostat applications, the only required modification is to change the PCB generated and navigate to the web URL link provided by Appliancizer. Finally, if the web application source code is modified, a simple page refresh allows the device to be updated.

Since our custom works like a browser for smart devices, users can navigate to smart devices web applications created by other developers and easily test their functionality after the PCB for the corresponding web application is replaced.

8.5 Limitations

Our tool has several limitations including limitations of the auto-routing algorithms which limit the complexity of supported PCB layouts and the compulsory use of JavaScript which may limit real-time performance. However, we narrow the discussion to the limitations presented in our essential interface mapping technique:

8.5.1 Limited HTML controls. Appliancizer supports a limited number of HTML elements that map to tangible interfaces. HTML radio buttons, checkboxes, lists, and others could potentially be used to map tangible interfaces allowing more intuitive web-based design prototypes of smart devices. For a physical switch, a HTML element with changing text ON and OFF is well supported with our current implementation. However, a checkbox HTML element may provide a better fit when prototyping.

8.5.2 Expressing hardware components capabilities and limitations. Hardware components have different capabilities which make them not always fully match with their HTML element counterpart. In Figure 1 the motorized potentiometer and the temperature sensor both map to the HTML slider element. The HTML slider element's value can be set pragmatically. However, while the motorized potentiometer also can be set programmatically (with some latency) the temperature sensor component cannot. Therefore, designers must be aware that, for some hardware components, the behavior will not exactly match the HTML element behavior when the graphical interface is transformed into a tangible one.

8.5.3 Mapping limitations. A essential interface mapping that reaches the most number of hardware components allows creating more varieties of smart devices. However, our current essential interface mapping approach is limited for some hardware components.

Appliancizer is well fitted for hardware sensors and actuators that require low essential data transfers. Hardware components that require transferring intensive information such as SD Cards, Ethernet ICs, etc are not suitable for our tool.

Among the suitable hardware components, our tool also presents some limitations. For example, mapping hardware components that require multiple essential interfaces. An example of one of these hardware components is the inertial measurement unit (IMU) that reports acceleration in three different axes (e.g, X, Y Z). For each axis, a numeric range of values is produced. This requires more than one slider element to fully map the IMU essential interfaces.

9 FUTURE WORK

Our research can expand in multiple directions, below we address some that we consider are worth further investigation:

Smart devices without screens Our tool targets smart devices that use displays. However, we see our implementation also useful for providing rapid designs of smart devices with virtual GUIs that could be accessed from smartphones or VR headsets.

Essential interface mapping Hardware components have different capabilities that make them unique. By further exploring their essential interfaces and how they interact with the software application will allow the creation of better mappings between graphical and tangible interfaces. Consequently, allowing the creation of more diverse smart devices and not limited by a few hardware components.

Web developers and hardware design Our tool gives web developers the potential ability to design smart devices. However, although the development is reduced to only selecting the hardware components for the design, web developers must still know the basic functionality of the hardware components used for the smart device. A further user study is required to validate if web developers with no previous hardware skills can create modern hardware designs.

Mixed interface access to a web application With our essential interface mapping technique the same web application can be accessed through a fully graphical or a mix of graphical and tangible interfaces. Use cases for this observed capability may exist in modern applications that require both modes of interaction, in our work, this remains unexplored.

10 CONCLUSION

This paper presented Appliancizer and essential interface mapping, a computational design tool and a novel technique that allows designers to rapidly generate functional smart devices from web-based prototypes, facilitating the prototyping of mixed graphical-tangible interactions. Our essential interface mapping technique maps the similarities between HTML controls and tangible controls which enables circuit synthesis and low-level code generation. This allows the transformation of HTML elements from graphical to tangible and enables the almost complete automation of the conventional development process for smart devices.

Appliancizer makes graphical and tangible interaction prototyping as easy as modifying a web page and allowing designers to leverage the well-developed ecosystem of web technologies. This accelerates development and reduces complexity for designers. We have demonstrated the capabilities of our tool and novel techniques by transforming both a video player and a smart thermostat web application into physical devices without requiring any changes to the application source code. We have also demonstrated the rapid development capabilities of our tool through a preliminary user study.

ACKNOWLEDGMENTS

The authors would like to thanks Dr. Scott Klemmer for his invaluable advice regarding the presentation of this work.

REFERENCES

[1] Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2017. Trigger-Action-Circuits: Leveraging Generative Design to Enable Novices to Design and Build Circuitry. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. ACM, 331–342.

[2] Arduino. 2020. Arduino Reference. https://www.arduino.cc/reference/en/

[3] Jonathan Bachrach, David Biancolin, Austin Buchan, Duncan W Haldane, and Richard Lin. 2016. Jitpcb. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. IEEE, 2230–2236.

[4] Rafael Ballagas, Meredith Ringel, Maureen Stone, and Jan Borchers. 2003. iStuff: a physical user interface toolkit for ubiquitous computing environments. In *Proceedings of the SIGCHI conference on Human factors in computing systems*. 537–544.

[5] Hernando Barragán. 2004. Wiring: Prototyping physical interaction design. *Interaction Design Institute, Ivrea, Italy* (2004).

[6] Ayah Bdeir. 2009. Electronics As Material: LittleBits. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction* (Cambridge, United Kingdom) *(TEI '09)*. ACM, New York, NY, USA, 397–400. https://doi.org/10.1145/1517664.1517743

[7] Jean-Pierre Charras. 2012. " Kicad: GPL PCB Suite".

[8] CodePen. 2020. Build, Test, and Discover Front-end Code - Nest Thermostat Control. https://codepen.io/dalhundal/pen/KpabZB

[9] Adrien Coyette and Jean Vanderdonckt. 2010. Prototyping Digital, Physical, and Mixed User Interfaces by Sketching. In *Workshop on User Interface eXtensible Markup Language UsiXML, France, Paris*.

[10] Flavia C Delicato, Paulo F Pires, Thais Batista, Everton Cavalcante, Bruno Costa, and Thomaz Barros. 2013. Towards an IoT ecosystem. In *Proceedings of the First International Workshop on Software Engineering for Systems-of-Systems*. 25–28.

[11] EAGLE. 2020. PCB Design Software - Autodesk. https://www.autodesk.com/products/eagle/overview

[12] EETimes. Apr. 2017. 2017 Embedded Markets Study: Integrating IoT and Advanced Technology Designs, Application Development Processing Enviroments. https://m.eet.com/media/1246048/2017-embedded-market-study.pdf.

[13] Jorge Garza, Devon J Merrill, and Steven Swanson. 2019. Amalgam: Hardware Hacking for Web Developers with Style (Sheets). In *International Conference on Web Engineering*. Springer, 315–330.

[14] Julien Gascon-Samson, Kumseok Jung, Shivanshu Goyal, Armin Rezaiean-Asel, and Karthik Pattabiraman. 2018. Thingsmigrate: Platform-independent migration of stateful javascript iot applications. In *32nd European Conference on Object-Oriented Programming (ECOOP 2018)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.

[15] Saul Greenberg and Chester Fitchett. 2001. Phidgets: Easy Development of Physical Interfaces Through Physical Widgets. In *Proceedings of the 14th Annual ACM Symposium on User Interface Software and Technology* (Orlando, Florida) *(UIST '01)*. ACM, New York, NY, USA, 209–218. https://doi.org/10.1145/502348.502388

[16] Grove. 2020. Grove modules - Seeed Studio. https://www.seeedstudio.com/category/Grove-c-1003.html

[17] Gaoyang Guan, Wei Dong, Yi Gao, Kaibo Fu, and Zhihao Cheng. 2017. TinyLink: A Holistic System for Rapid Development of IoT Applications. In *Proceedings of the 23rd Annual International Conference on Mobile Computing and Networking*. ACM, 383–395.

[18] Bjoern Hartmann, Scott R Klemmer, Michael Bernstein, and Nirav Mehta. 2005. d. tools: Visually prototyping physical UIs through statecharts. In *in Extended Abstracts of UIST 2005*. Citeseer.

[19] Utkarshani Jaimini and Mayur Dhaniwala. 2016. JavaScript empowered Internet of things. In *2016 3rd International Conference on Computing for Sustainable Global Development (INDIACom)*. IEEE, 2373–2377.

[20] JLCPCB. 2020. Low Cost PCB Prototype & PCB Fabrication. https://jlcpcb.com/

[21] Jun Kato and Masataka Goto. 2017. F3. js: A parametric design tool for physical computing devices for both interaction designers and end-users. In *Proceedings of the 2017 Conference on Designing Interactive Systems*. 1099–1110.

[22] Majeed Kazemitabaar, Jason McPeak, Alexander Jiao, Liang He, Thomas Outing, and Jon E. Froehlich. 2017. MakerWear: A Tangible Approach to Interactive Wearable Creation for Children. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. ACM, New York, NY, USA, 133–145. https://doi.org/10.1145/3025453.3025887

[23] Mannu Lambrichts, Jose Maria Tijerina, and Raf Ramakers. 2020. SoftMod: A Soft Modular Plug-and-Play Kit for Prototyping Electronic Systems. In *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*. 287–298.

[24] Richard Lin, Rohit Ramesh, Antonio Iannopollo, Alberto Sangiovanni Vincentelli, Prabal Dutta, Elad Alon, and Björn Hartmann. 2019. Beyond Schematic Capture: Meaningful Abstractions for Better Electronics Design Tools. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. 1–13.

[25] Victor Matos. 2009. Android Environment Emulator. *Cleveland State University* 20, 11 (2009).

[26] David A Mellis and Leah Buechley. 2014. Do-it-yourself cellphones: an investigation into the possibilities and limits of high-tech diy. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. 1723–1732.

[27] David A Mellis, Leah Buechley, Mitchel Resnick, and Björn Hartmann. 2016. Engaging amateurs in the design, fabrication, and assembly of electronic devices. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*. ACM, 1270–1281.

[28] Devon J Merrill, Jorge Garza, and Steven Swanson. 2019. Echidna: mixed-domain computational implementation via decision trees. In *Proceedings of the ACM Symposium on Computational Fabrication*. 1–12.

[29] Multisim. 2020. Live Online Circuit Simulator. https://www.multisim.com/

[30] Zainalabedin Navabi. 1997. *VHDL: Analysis and modeling of digital systems*. McGraw-Hill, Inc.

[31] Mozilla Developer Network. 2013. Firefox OS, 2013.

[32] P. Li and J. Bachrach, "The Stanza Language". 2015. http://www.stanzalang.org

[33] Raspberry Pi. 2015. Teach, Learn, and Make with Raspberry Pi. https://www.raspberrypi.org/

[34] Proteus. 2020. PCB Design and Circuit Simulator Software. https://www.labcenter.com/

[35] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. Retrofab: A design tool for retrofitting physical interfaces using actuators, sensors and 3d printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. 409–419.

[36] Rohit Ramesh, Richard Lin, Antonio Iannopollo, Alberto Sangiovanni-Vincentelli, Björn Hartmann, and Prabal Dutta. 2017. Turning coders into makers: the promise of embedded design generation. In *Proceedings of the 1st Annual ACM Symposium on Computational Fabrication*. ACM, 4.

[37] Valkyrie Savage, Sean Follmer, Jingyi Li, and Björn Hartmann. 2015. Makers' Marks: Physical markup for designing and fabricating functional objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*. 103–108.

[38] Sunil Saxena and HV Kwon. 2012. Tizen architecture. In *Tizen Developer Conference, San Francisco, California*.

[39] Swoop. 2015. A Python Library for Eagle PCB Design Files. https://github.com/NVSL/Swoop/

[40] Nest Learning Thermostat. 2011. v1. 0 [Product Brochure]. *Nest Labs, plus publication date addendum* (2011). https://nest.com/

[41] Donald Thomas and Philip Moorby. 2008. *The Verilog® Hardware Description Language*. Springer Science & Business Media.

[42] TinkerCad Circuits. 2020. https://www.tinkercad.com/learn/circuits

[43] Blase Ur, Elyse McManus, Melwyn Pak Yong Ho, and Michael L. Littman. 2014. Practical Trigger-action Programming in the Smart Home. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems* (Toronto, Ontario, Canada) *(CHI '14)*. ACM, New York, NY, USA, 803–812. https://doi.org/10.1145/2556288.2557420

[44] Nicolas Villar, James Scott, Steve Hodges, Kerry Hammil, and Colin Miller. 2012. . NET gadgeteer: a platform for custom devices. In *International Conference on Pervasive Computing*. Springer, 216–233.

[45] Webcomponents. 2020. Specifications. https://www.webcomponents.org/specs

[46] YouTube Player API Reference for iframe Embeds - Google Developers. 2020. https://developers.google.com/youtube/iframe_api_reference