Jacdac-for-Max: Plug-and-Play Physical Prototyping of Musical Interfaces

Kobi Hartley
Lancaster University
UK
k.hartley1@lancaster.ac.uk

Steve Hodges
Microsoft Research
UK
shodges@microsoft.com

Joe Finney
Lancaster University
UK
j.finney@lancaster.ac.uk

ABSTRACT

This article presents Jacdac-for-Max: a cross-platform, open-source set of node.js scripts and custom Cycling '74 Max objects which enable the use of Jacdac, an open, modular plug-and-play hardware prototyping platform, with Max visual programming language frequently used for audio-visual applications. We discuss the design and implementation of Jacdac-for-Max, and explore a number of example applications. Through this we show how Jacdac-for-Max can be used to rapidly prototype digital musical interfaces based on a range of input devices. Additionally, we discuss these qualities within the context of established principles for designing musical hardware, and the emerging concepts of long-tail hardware and frugal innovation. We believe that through Jacdac-for-Max, Jacdac provides a compelling approach to prototyping musical interfaces while supporting the evolution beyond a prototype with more robust and scalable solutions.

Author Keywords

NIME, musical hardware, prototyping toolkits, long-tail hardware

CCS Concepts

•**Applied computing** → **Sound and music computing;** •**Hardware** → **PCB design and layout;** •**Human-centered computing** → *User interface toolkits;*

1. INTRODUCTION

Artists searching for new forms of musical expression often turn to creating their own instruments. But creating a new instrument can involve a complex, time-consuming and costly journey. Electronic prototyping platforms can provide a more accessible route, and using these has become easier with platforms such as LittleBits [5], Arduino [1], Bela [28] and Raspberry Pi [32]. Indeed, the aforementioned platforms have all been successfully used for developing new interfaces that support musical expression. However, working with electronics still requires technical knowledge of components, circuit design and firmware—all of which present barriers to less experienced users. Coupled with this, users wishing to transition beyond the prototype face additional complexities of redesigning devices for manufacture, resulting in a large number of devices which never realise their true impact. These issues have been discussed most notably within the context of long-tail hardware [20, 21], but similar themes and challenges have been highlighted within NIME-related work [14, 18, 19]. Thus, there are shared demands for approaches to developing NIMEs which can address at least some of the barriers to scale devices beyond the prototype stages.

Jacdac [16] is a recently-developed modular plug-and-play prototyping platform that removes some of the complexity of prototyping through both hardware and software abstraction. Furthermore, Jacdac modules can enforce design standards with manufacturability in mind [2], and these qualities have been shown to make Jacdac a suitable platform to evolve devices beyond the prototype stages [17].

This paper describes Jacdac-for-Max: a cross-platform Max package that is closely coupled to Jacdac's service specification and which extends Jacdac's modular, plug-and-play properties. It allows users to map events or values received from Jacdac modules to other Max objects. We explore the various forms devices can take by way of three examples, which we discuss through three contexts: established design principles for musical hardware controllers [19], long-tail hardware [21] and frugal innovation [3]. We highlight qualities common to these contexts and reflect on how Jacdac-for-Max, coupled with Jacdac, presents a new route for artists to create and adapt instruments. Additionally, we explore Jacdac's suitability to evolve devices beyond prototypes [17], drawing attention to how issues associated with long-tail hardware [20, 21] are related to contemporary design guidelines for musical controllers [19].

2. RELATED WORK & MOTIVATION

This review of related work first considers embedded electronics, hardware prototyping toolkits and techniques, and in particular discusses design principles for musical contexts. Secondly, there is relevant work covering the mapping of controllers to sound, exploring common frameworks and popular methods, including some used in Max. For clarity, we also provide a subsection describing the Jacdac electronics prototyping platform and ways in which it can be programmed. Lastly, we present the motivation of this work.

2.1 Musical Interface Hardware Exploration

Embedded hardware has become increasingly accessible over recent years with the rising popularity of hardware proto-

NIME'23, 31 May–2 June, 2023, Mexico City, Mexico.

typing tools such as Raspberry Pi [32], Arduino, and many other increasingly modular approaches such as Adafruit's STEMMA and Sparkfun's Qwiic [6], as well as many others [26]. As such, we see embedded hardware being used for many different creative purposes, including music-making, as exemplified by Bela [28], an embedded hardware platform for low-latency control of sound. As well as Korg's collaboration with littleBits [31] a collection of analog hardware modules that can be linked together to construct new instruments and interfaces. Probatio [10] is similar, open source, toolkit that allows users to design and assemble their own digital musical instruments from discrete blocks. Other approaches such as BlockJam [30], ReacTable [24] and MO [34] further explore modularity to encourage ad-hoc construction of musical artefacts. Additionally, additive manufacturing approaches have been embraced to embed hardware into 3D printed structures which can be used to rapidly prototype, or augment existing instruments [11].

However, many of these prototyping approaches have drawbacks such as: the complexity of initial setup for less experienced users, the required system configuration knowledge when adjusting the hardware, and limits to the number of modules that may be connected at once. Moreover, while getting to an initial prototype is achievable for most—transitioning beyond this prototype to a device suitable for low-volume deployment is extremely difficult [20, 21]. These difficulties can relate to the complexity and cost associated with the need to completely redesign a device from prototype to deployment, identifying and sourcing components as well as manufacturing considerations. Some of these issues have been discussed within the context of musical interfaces, mainly relating to issues of longevity in NIMEs [27, 29], resulting in a large number of DMIs (digital musical interface) being abandoned soon after they are created. These concerns have been underscored in work by Hattwick and Wanderley [19], where they provide a set of principles, reinterpreted from those originally proposed by Perry Cook [12, 13]. Hattwick and Wanderley's principles shift the focus to, among other things, *robustness, re-usability* and *manufacturability.*

When considered in the context of barriers of long-tail hardware [20, 21] and moving beyond the prototype [25], there are clear, shared shortcomings preventing large numbers of musical interface prototypes from ever realising their true impact.

2.2 Mapping Controllers to Sound

The mapping of different types of physical expression to parameters of the music generation process and other forms of control is a central tenet of musical interface design [23]. There are different forms of mapping such as one-to-one, one-to-many and many-to-many, where these define the relationship between the physical control(s) and associated parameter(s) [39]. In some cases, it is useful to create intermediate mappings, to support additional processing between controls and the resulting sounds [22].

There are many applications which showcase the value of creating custom mappings. For example, an Ableton-related project Mapper4Live [8], uses the Max-for-Live framework to expose Ableton's synthesis and control engines, to allow users to map forms of interaction between controllers and hardware/software synthesis. Mapping of sound to hardware has also been shown to assist the visually impaired in prototyping new devices [33].

Max has been used frequently, either to map external hardware to elements in Max, such as mira [37] which facilitates control over elements in Max using a connected iPad. Similarly Max has been used to program and control hardware, a prominent example being Oopsy [38] which uses Max/gen∼, along with an Arduino to program Eurorack modular hardware. As well as the previously mentioned MO [34], which presents both Arduino and Max as programming routes. Max's visual interface embodies this mapping metaphor with objects connected together via patch cords. Many software packages have built upon Max's mapping features like MnM [7] which allows users to map gestures to forms of sound control. These examples demonstrate Max's pragmatism for mapping, both with respect to the programming approach it encourages as well as the support for varied external hardware.

2.3 Jacdac

Jacdac [16], an open-source, modular, plug-and-play device creation platform, addresses drawbacks associated with some of the established hardware prototyping systems. Jacdac modules in the form of sensors, actuators, display elements etc., are coupled with a service-based model [4] that provides abstractions over the hardware. This reduces complexity in programming and debugging. In addition, Jacdac has been designed to support the construction of devices that are more robust than a typical electronics prototype [2]. This has been explored in MakeDevice, a web-based tool which allows users to generate custom carrier circuit boards for permanently mounting Jacdac modules, and to design enclosures for the resulting Jacdac devices [17].

Options for programming Jacdac include MakeCode [15], a popular web-based drag-and-drop programming approach which uses visual code blocks. Other options include Python and TypeScript. If users want to explore crafting novel musical controllers for Max, this would require either writing Max externals in C or an implementation of the Jacdac TypeScript stack (Jacdac-ts) within Max's node-for-max framework.

2.4 Motivation

The core motivation for this work is addressing issues related to prototyping and developing new musical interfaces with embedded hardware.

We aim to build on previously-demonstrated physical tools for musical expression [9, 10, 30, 34], by addressing the need for higher fidelity prototypes [14], as well as consideration of how promising new solutions can be produced beyond one-off prototypes to support adoption at scale [18, 19, 27, 29]. We believe that a route to addressing this gap is to leverage the scalability features of Jacdac [2] and thereby address the replication challenge of long-tail hardware [21].

Jacdac's modular, plug-and-play nature has advantages compared to other embedded hardware toolkits [16], coupled with previous work exploring Jacdac beyond the prototype [17]. We propose to extend Jacdac for use within musical contexts, while maintaining and building upon its modular, plug-and-play properties. Given the established nature of Max within the creative community, and the flexibility it provides in terms of objects and their mapping, we believe this integration is a fruitful path of exploration.

3. DESIGN & IMPLEMENTATION

In this section we detail the design and implementation of Jacdac-for-Max. This includes an overview of the architecture and frameworks used, the node-for-max scripts, and the Max objects we've created to represent each of the different Jacdac services. Additionally, we detail how our implementation automatically generates node.js snippets and the required Max objects directly from the Jacdac service specification.

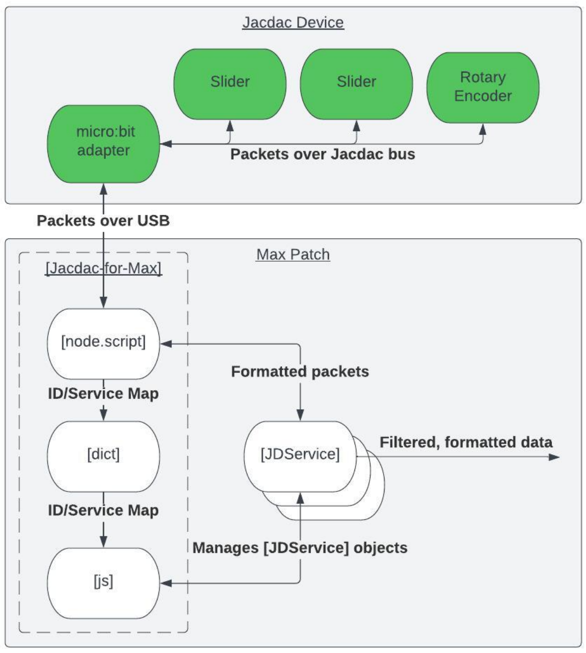

Figure 1: Architecture of Jacdac-for-Max. A [node.script] object connects to and receives packets from the Jacdac Bus, formats these and then sends them to all [JDService] objects. These filter the relevant data, and route it to specified outlets.

3.1 Jacdac-for-Max Object

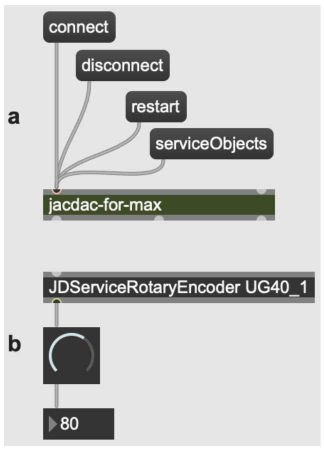

Figure 2: Example of how Jacdac-for-Max is used within a patch. (a) A [jacdac-for-max] object with some common commands used to interface with it. (b) A [JDServiceRotaryEncoder] object which serves as an end-point for data from a connected Jacdac slider.

The [jacdac-for-max] object is the first object the user will instantiate. As shown in Figure 1, it contains: the node.js scripts for connecting and processing messages from the Jacdac bus, a [dict] object storing a map of connected devices and a [js] object for creating/deleting [JDService]

```
// get (register to REPORT_UPDATE event to enable background
      refresh)
const positionReg = service.register(RotaryEncoderReg.
      Position);
positionReg.on(REPORT_UPDATE, () => {
    const [position] = positionReg.unpackedValue;
    maxApi.outlet([service.maxID.toString(), "position",
         position.toString()]);
});
```

Figure 3: A TypeScript code snippet that shows how event listeners are generated for the position register of the rotary encoder service.

objects. It can receive a number of commands to connect/disconnect or restart the Jacdac connection, as well as generate [JDService] objects (see Figure 2a).

3.2 Jacdac and Node-for-Max

A node implementation of Jacdac-ts presents a straightforward route to expose the Jacdac bus in the node-for-max framework. Jacdac-for-Max leverages Jacdac's service model [4], resulting in a total of 39 sensor-based supported services. Each Jacdac sensor service is represented as a function within the [node.script] object seen in Figure 1.

These functions all follow a similar design pattern: creating an event listener for any register or event changes, formatting these event messages as arrays of strings, and adding the ID specific to that service instance. The result is emitted from the [node.script] objects outlet. Figure 3 provides an example function for reporting updates to the position register of a rotary encoder service. All messages are sent using Max's [send] object and each [JDService] receives all messages.

3.3 Max Objects for Jacdac Services

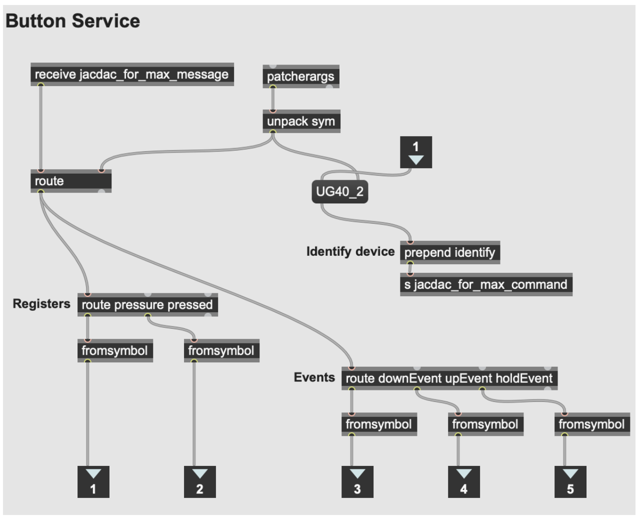

Figure 4: Jacdac Button service patch, showing how messages for register values and events are routed and directed to individual outlets.

3.3.1 Structure of JDService objects

For each of the 39 supported Jacdac services, there is a Max object to represent it. These all follow the naming convention of [JDServiceX], where X is the service's full name; these are represented as [JDService] in Figure 1. An example of how these objects are instantiated within a Max patch can be seen in Figure 2. Additionally, Figure 4 shows the internal sub-patch of a Max object for a Jacdac button service; this conforms tightly to the service specification, exposing read-only registers specific to that service.

3.3.2 Filtering received messages

As multiple connected devices can provide the same service, we create Max objects for each unique device and service combination. This is achieved by passing the *friendly name* of a service as a parameter into a Jacdac-for-Max service object. In Figure 2(b) the [JDServiceRotaryEncoder] object has the parameter 'UG40_2', and any messages received without that ID are not processed by the given JDService object.

3.3.3 Spawning or removing JDService objects

To promote plug-and-play with Jacdac-for-Max, we provide users with the ability to automatically spawn [JDService] objects for all of their connected modules. To keep track of which services and modules are present, a [dict] Max object stores a mapping between the *friendlyName* of a service and its full name. This updates automatically as the state of the Jacdac bus updates, see Figure 5 for an example.

When the user requests to generate [JDService] objects, these values are passed a Max [js] object, which will create or delete [JDService] objects within the main Max patch. The full service name stored in the [dict] object is used to identify which [JDService] object to create, and the *friendlyName* is passed as a parameter to that object.

```
{
 UG40_0 : control,
 UG40_1 : rotary encoder,
 UG40_2 : button
}
```

Figure 5: A dictionary stores the mappings between services' friendly names (unique identifiers) and the corresponding long names.

3.4 Generation from Service Specification

Both the functions used to create event listeners for events and registers (section 3.2), and the JDService Max objects (section 3.3), follow consistent design patterns. This allows Jacdac-for-Max to generate all of these objects directly from the Jacdac service specification. This is done primarily to promote a strong coupling between Jacdac-for-Max implementation and the Jacdac service specification. This means that few, if any, changes are required to Jacdac-for-Max's implementation when new Jacdac sensor services are created. Additionally, this supports maintainability of the code, reducing the points at which errors could arise, compared to the manual implementation of services.

3.5 Performance

Jacdac's performance has been previously documented [4], as such we'll draw heavily from this work. One aspect of Jacdac highlighted by this work is it's intended use, that is: *"to create an embedded system from a small network of low-bandwidth sensors and actuators, with one to two handfuls of devices (modules and brain)"* [4]. So while latency may not be as low when compared to other hardware, robustness and ease of implementation is prioritised. That being said, Ball et al's 2023 work includes information key for NIME contexts. With respect to *latency*: the time between a Jacdac module sending a packet and it being received is under 500μs. In relation to *streaming intervals*, sensor stream data is the largest user of wire time, so the advised streaming interval for Jacdac is no more than 50Hz.

While an evaluation of Jacdac-for-Max has not yet taken place, we have conducted preliminary measurements of gesture-to-sound latency with Jacdac-for-Max. For our experimental setup we chose to emulate a previous approach [40]. We created a simple Jacdac-for-Max patch, where a Jacdac button press was used to trigger a click audio impulse, the audio was recorded through an external microphone into a different computer. The Jacdac brain used was an RP2040, connected via USB-C over serial connection, the computer used was a 2020 MacBook with an M1 processor and 16GB of RAM. To calculate latency we measured the time between the sound the button press and the click sound generated in Max. To provide a comparison, we did the same test but instead a keyboard press which triggered the click. We repeated this 10 times in order to generate an average. In our setup Jacdac-for-Max had an average gesture-to-sound latency of 47ms, with a range of 40-60ms, for comparison the latency from a keyboard key press was an average of 38ms with a range of 30-50ms. It is worth noting our experimental setup does not account for additional hardware latencies. From the results, Jacdac-for-Max adds an average gesture-to-sound latency of 9ms.

While these results are preliminary, a wide-ranging analysis of performance and user evaluation is planned for future work, but the results presented here should provide an adequate reference point. We expect that the performance of Jacdac-for-Max will be worse compared to lower level approaches such as Bela [28], with the trade-off being the modularity and service-level abstraction providing an easier entry point to novices who wish to prototype musical interfaces.

4. EXPLORING JACDAC-FOR-MAX

To demonstrate the range of applications made possible through Jacdac-for-Max, we present 3 examples. We have selected these to express the range in both functionality that Jacdac-for-Max unlocks, as well as the form the resulting interfaces can take.

4.1 Wired Prototyping

We start with a simple buffer sampler with controls for changing the buffer start position, sample length, pitch and volume. We've used Jacdac-for-Max to map these controls to a number of Jacdac modules. You can see a Figure of the patch in presentation mode in Figure 6. We show here how values received from JDService objects, can be scaled and used within a Max patch.

In Figure 7, we show a set of wired Jacdac modules which are used to control various aspects of the simple buffer sampler patch shown in Figure 6. This form of Jacdac is the simplest and most common, where modules are connected together using Jacdac cables. This maximizes flexibility, allowing users to plug or unplug modules when iterating on a prototype.

4.2 Evolving Beyond the Prototype

Previous work has discussed the barriers faced when moving beyond a desktop prototype, to a more robust device suitable for deployment [25]. This is especially true when scal-

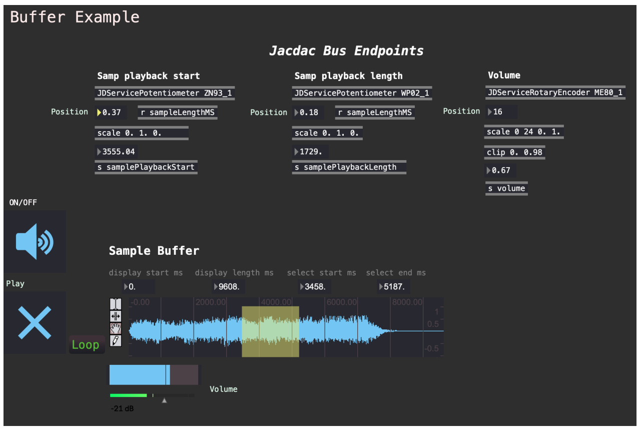

Figure 6: Buffer example Max patch, showing how data received from Jacdac modules can be scaled and mapped to other parameters.

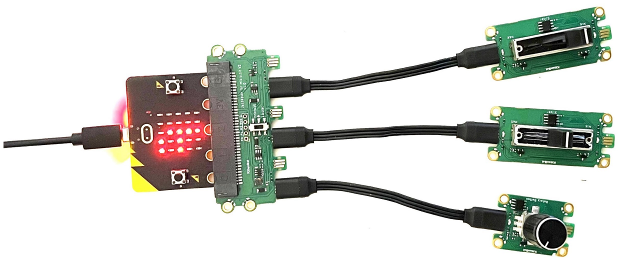

Figure 7: A wired Jacdac device. Modules here are used to control the parameters of the buffer sampler seen in Figure 6. The modules pictured are: a Jacdac micro:bit adapter connected to a micro:bit, a button/rotary encoder, and two sliders.

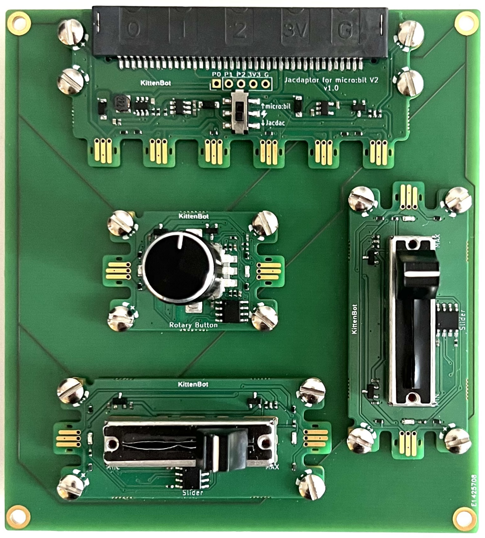

Figure 8: An automatically-designed carrier board with Jacdac modules screwed to it. There are two Jacdac sliders, a rotary control and an adapter for connecting a micro:bit (not shown).

ing to tens or more copies of a prototype device, when cost starts to be a concern too. A common approach involves the transition to a custom-designed circuit, but this can require a lot of time, skill and money. Elements of Jacdac have been designed with overcoming these barriers in mind. For example, the module mounting holes not only support physical mounting, but also provide a way to connect modules electrically as an alternative to using Jacdac cables. This makes it possible to mount Jacdac modules onto a 'carrier' circuit board that provides electrical connectivity between modules using traces, as explored by MakeDevice [17]. This work also demonstrates how enclosures for the resulting Jacdac devices can be generated semi-automatically to further facilitate deployment.

The resulting device is of a consistent, often smaller form factor. The features described here allow for exploration of devices in higher fidelities, with more robust and compact construction. This has not only been discussed as a core need in research [14, 18], but is also instrumental in live performance contexts [36].

In Figure 8 we show how Jacdac modules can be arranged on and screwed to a carrier board, eliminating the need for cables. The figure shows the same device detailed in Section 4.1—with two sliders and a rotary control to modulate aspects of a buffer sampler—implemented in this more deployable form.

4.3 Augmenting Instruments

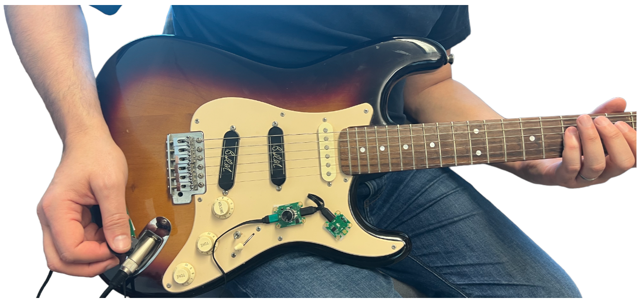

Figure 9: An electric guitar augmented with a Jacdac slider, rotary control and light sensor. The person pictured is using the Jacdac slider on the left to modulate the dry/wet mix of a delay effect in Max.

The final example we include here is how Jacdac modules, along with Jacdac-for-Max can be used to augment an existing instrument. We show in 9, a guitar with slider, rotary control and light sensor modules attached by velcro, and connected together using Jacdac wires. In this instance, they are used to control a series of effects within Max. The rotary and slider modules are used to control delay effect times and amounts. The light sensor is used to modulate, with a hand, the amount of tremolo, acting as an analogue to a traditional tremolo or 'whammy' bar.

This example can be extended in a similar way to the example shown in Section 4.2. Instead of attaching modules to the front of an instrument and connecting them via Jacdac cables, they could be mounted onto a carrier PCB and placed inside the instrument.

5. DISCUSSION

In this section we discuss the wider values of Jacdac and Jacdac-for-Max. We reflect on the work presented here within the context of established design principles for musical hardware and long-tail hardware. Finally, we discuss our work in relation to this year's conference theme of frugal innovation.

5.1 Unlocking a Long Tail of Musical Interfaces

While new prototyping methods and tools for new interfaces for musical expression have previously been demonstrated [9, 10], there is still a need for exploration of higher fidelity prototypes [14] as well as consideration for the longevity and replication [27, 29].

As discussed in the motivation (Section 2.4) and mentioned again with our examples (Section 4.2), developers of new musical interfaces can face significant barriers when making this transition from a hardware prototype to something suitable for deployment or replication [25]. Beyond this, further manufacturing considerations may need to be taken into account to support adoption at scale [18, 19]. These barriers are shared with those identified in previous work on the concept of long-tail hardware [21]. Indeed, a number of key design principles outlined in previous work but underscored by Hattwick and Wanderley are related to issues associated with long-tail hardware—in particular the principles of *re-usability*, *robustness* and *manufacturability*. Here we will discuss how Jacdac and Jacdac-for-Max can address these.

Firstly, in relation to *re-usability*, Jacdac's modularity and service-based representation [4] means that rotary modules from different manufacturers will behave equivalently, as long as they present the same Jacdac *service*. This means that people with Jacdac modules from different manufacturers can easily replicate each others devices. This re-usability extends to Jacdac-for-Max too; because the Max objects provided are strongly coupled to the Jacdac service model, a patch shared between users will work seamlessly. In this way, we promote Jacdac and Jacdac-for-Max as a way for users to easily share and collaborate on creative devices, thereby removing the complexity associated with the today's frequent need to replicate exact hardware implementations. However, Jacdac in its wired desktop form, if orientation and position of modules is key to a device, re-usability may be hampered. But using concepts discussed in 4.2 and outlined in previous work [17], modules can be screwed down to a custom carrier board. In this way, a consistency of orientation and position of modules is enforced, ensuring *re-usability* and replicability of a device. These points also relate to supporting *robustness*, not just of devices in the prototyping stage, but also beyond it. Jacdac modules, specifically those supporting the EC30 form factor [2], make it easier to design enclosures for Jacdac devices. Such that devices can be designed to withstand shock or harsh conditions (e.g. in a live performance setting). These aspects additionally inform *manufacturability*, with Jacdac's EC30 form factor [2] promoting surface mounting of modules to carrier PCBs as shown with MakeDevice [17], and in Section 4.2. While methods for transitioning Jacdac to a form which prioritises *manufacturability* and *scalability* is in it's early stages, there is scope to use Jacdac as a vehicle to explore the flattening of individual modules onto a single PCB, removing redundant components and generating a new but equivalent PCB design which can be easily manufactured at scale.

5.2 Frugal Innovation

Considering the theme of NIME 2023, we thought it valuable to reflect on how Jacdac-for-Max can lower, but also present barriers in using Jacdac for frugal music innovation. We do this with particular reference to the core competencies outlined by Santa Clara University's Frugal Innovation Hub [3].

Firstly, Jacdac's protocol supports implementation on relatively low-priced micro-controllers [16], resulting in a end price which is *affordable* and certainly comparable to other modular prototyping platforms, if not lower. Although the use of one MCU per sensor does have a negative impact on overall cost. Additionally, Jacdac-for-Max's reliance on proprietary, paid software (Max) does present a barrier to use, although Jacdac-for-Max as a package will be released for free and open source. Jacdac, with its modular form and intuitive plug-and-play approach is *human-centric*, in that it builds on previous tools and work showing that modularity fosters creativity in the crafting process [35]. Clear roles and services of modules, present a *simplicity* in creating multi-functional devices, as has been reinforced by a previous evaluation of Jacdac [16]. Jacdac-for-Max embodies this model, providing a straightforward way for users to connect and use Jacdac modules within Max. Additionally, by exposing individual service/device combinations as Max objects, we conform both to Jacdac's service model and typical Max practices. This also brings a level of high *adaptability*, both in how users can chain together physical Jacdac modules, and how they can map these values to various Max objects. Jacdac modules are *lightweight* and *mobile*, as shown in Section 4.3 and great for augmenting existing instruments. Albeit, Jacdac-for-Max's requirement for a computer running Max can reduce its suitability in these, as well as *rugged* contexts. However, as shown in the example in Section 4.2, and discussed further in the context long-tail hardware in Section 2.4: Jacdac's form factor and design choices make assembly with carrier PCBs, design for enclosures easier to achieve [17]. These aspects could be extended further to include IP rated enclosures, more suitable for rugged deployments. It is worth noting another aspect of Jacdac which go against the qualities of frugal innovation. that having one MCU per sensor is by no means a *green* approach to developing hardware.

6. FUTURE WORK & CONCLUSION

As for future work, we wish to build and integrate a wider range of musical interfaces based on Jacdac, as well as increase the number of sensors and actuators which can be used with Jacdac-for-Max. We would also like to explore further how Jacdac-for-Max can be used to ease the creation of novel musical controllers and their integration into enclosures. We plan to explore these concepts further, co-designing with artists to evolve a Jacdac prototype, which uses Jacdac-for-Max, into forms suitable for low-volume deployment and use in performance. Additionally, in November 2022, Cycling '74 announced RNBO[1], a new Max-like workflow, which lets users to export RNBO patches to targets including VST, Web Assembly and Raspberry Pi. Given RNBO is unlocking possibilities for how hardware can be used in musical interfaces, we are interested in extending RNBO to support Jacdac as a compilation target. While the current selection of Jacdac brains don't offer the processing capabilities to support RNBO, there is scope for developing a Jacdac brain which can. This would allow a Jacdac solution which combines Jacdac modules like sliders and rotary controls with a Jacdac audio output module and a Jacdac processor module configured using Jacdac-for-Max, would remove the need for a computer, increasing

[1]cycling74.com/products/rnbo

deployment options.

In conclusion, this work has presented Jacdac-for-Max, a package which extends Jacdac's, modular, plug-and-play characteristics to Max. Allowing users to prototype and develop new forms of musical controllers within Max, without in-depth knowledge of embedded hardware. We have explored how users can construct devices using Jacdac, and program musical controllers using Jacdac-for-Max, by way of three examples. Additionally, we discussed the value this work holds in relation to established design principles [19], and drew parallels with work relating to the long-tail of hardware [21] and frugal innovation [3]. We have highlighted how characteristics of Jacdac, which are built upon with Jacdac-for-Max, might help artists in designing and prototyping devices. We believe the platforms presented make it easier for users to explore, among other things, *reusability*, *robustness* and *manufacturability*. We reflected on how these qualities help users in exploring devices at higher fidelities and in more robust forms, unlocking a long tail of musical interfaces.

7. NIME ETHICAL STATEMENT

This work was conducted in line with NIMEs ethical guidelines. Funding for this research was provided by the UK EPSRC, grant number: **EP/V519558/1**.

8. ACKNOWLEDGEMENTS

We would like to additionally thank Peli De Halleux, James Devine, Gabriele D'Amone, David Gakure, Tom Ball, Michal Moskal, Matt Oppenheim and Lorraine Underwood for their contribution to discussions around this work.

9. REFERENCES

[1] Arduino 2021, arduino homepage. `https://www.arduino.cc`. Accessed: 2021-09-13.

[2] EC30 Form Factor. `https://microsoft.github.io/jacdac-docs/ddk/design/ec30/`. Accessed: 2022-10-09.

[3] Frugal innovation hub, santa clara university | about us. `https://www.scu.edu/engineering/labs--research/labs/frugal-innovation-hub/about-us/`. Accessed: 2022-11-11.

[4] T. Ball, P. de Halleux, J. Devine, S. Hodges, and M. Moskal. Jacdac: Service-based prototyping of embedded systems. Technical Report MSR-TR-2023-4, Microsoft, January 2023.

[5] A. Bdeir. Electronics as material: Littlebits. In *Proceedings of the 3rd International Conference on Tangible and Embedded Interaction*, TEI '09, page 397–400, New York, NY, USA, 2009. Association for Computing Machinery.

[6] C. Bell. *Introducing Qwiic and STEMMA QT*, pages 217–258. Apress, Berkeley, CA, 2021.

[7] F. Bevilacqua, R. Müller, and N. Schnell. Mnm: a max/msp mapping toolbox. In *New Interfaces for Musical Expression*, pages 85–88, 2005.

[8] B. Boettcher, J. Malloch, J. Wang, and M. M. Wanderley. Mapper4live: Using Control Structures to Embed Complex Mapping Tools into Ableton Live. In *NIME 2022*, dec 13 2021. https://nime.pubpub.org/pub/lrhx3rpu.

[9] F. Calegario, M. Wanderley, S. Huot, G. Cabral, and G. Ramalho. A method and toolkit for digital musical instruments: Generating ideas and prototypes. *IEEE MultiMedia*, 24:63–71, 02 2017.

[10] F. Calegario, M. Wanderley, J. Tragtenberg, J. Wang, J. Sullivan, E. Meneses, I. Franco, M. Kirkegaard, M. Bredholt, and J. Rohs. Probatio 1.0: collaborative development of a toolkit for functional dmi prototypes. In *Proceedings of the international conference on New Interfaces for Musical Expression*, pages 21–25, 2020.

[11] D. Cavdir. Embedding electronics in additive manufacturing for digital musical instrument design. In *Proceedings of the Proceedings of the Sound and Music Computing Conferences*, 2020.

[12] P. Cook. 2001: Principles for designing computer music controllers. In *A NIME Reader*, pages 1–13. Springer, 2017.

[13] P. R. Cook. Re-designing principles for computer music controllers: a case study of squeezevox maggie. In *NIME*, volume 9, pages 218–221, 2009.

[14] L. Dahl. Designing new musical interfaces as research: What's the problem? *Leonardo (Oxford)*, 49(1):76–77, 2016.

[15] J. Devine, J. Finney, P. de Halleux, M. Moskal, T. Ball, and S. Hodges. MakeCode and CODAL: Intuitive and efficient embedded systems programming for education. *ACM SIGPLAN Notices*, 53(6):19–30, 2018.

[16] J. Devine, M. Moskal, P. de Halleux, T. Ball, S. Hodges, G. D'Amone, D. Gakure, J. Finney, L. Underwood, K. Hartley, et al. Plug-and-play physical computing with Jacdac. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 6(3):1–30, 2022.

[17] K. Hartley, J. Finney, S. Hodges, P. De Halleux, J. Devine, and G. D'Amone. Makedevice: Evolving devices beyond the prototype with jacdac. In *Proceedings of the Seventeenth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '23, New York, NY, USA, 2023. Association for Computing Machinery.

[18] I. Hattwick, J. W. Malloch, and M. M. Wanderley. Forming shapes to bodies: Design for manufacturing in the prosthetic instruments. In *NIME*, pages 443–448, 2014.

[19] I. Hattwick and M. M. Wanderley. Design of hardware systems for professional artistic applications. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 436–441, Copenhagen, Denmark, 2017. Aalborg University Copenhagen.

[20] S. Hodges. Democratizing the production of interactive hardware. In *UIST 2020*, October 2020.

[21] S. Hodges and N. Chen. Long tail hardware: Turning device concepts into viable low volume products. *IEEE Pervasive Computing*, 18(4):51–59, 2019.

[22] A. Hunt and M. M. Wanderley. Mapping performer parameters to synthesis engines. *Organised sound*, 7(2):97–108, 2002.

[23] A. Hunt, M. M. Wanderley, and M. Paradis. The importance of parameter mapping in electronic instrument design. *Journal of New Music Research*, 32(4):429–440, 2003.

[24] S. Jordà, G. Geiger, M. Alonso, and M. Kaltenbrunner. The reactable: exploring the synergy between live music performance and tabletop tangible interfaces. In *Proceedings of the 1st*

international conference on Tangible and embedded interaction, pages 139–146, 2007.

[25] R. Khurana and S. Hodges. Beyond the prototype: Understanding the challenge of scaling hardware device production. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–11, New York, NY, USA, 2020. Association for Computing Machinery.

[26] M. Lambrichts, R. Ramakers, S. Hodges, S. Coppers, and J. Devine. A survey and taxonomy of electronics toolkits for interactive and ubiquitous device prototyping. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(2), jun 2021.

[27] A. Marquez-Borbon and J. Martinez Avila. The problem of dmi adoption and longevity: Envisioning a nime performance pedagogy. 06 2018.

[28] A. McPherson. Bela: An embedded platform for low-latency feedback control of sound. *The Journal of the Acoustical Society of America*, 141(5):3618–3618, 2017.

[29] F. Morreale et al. Design for longevity: Ongoing use of instruments from nime 2010-14. 2017.

[30] H. Newton-Dunn, H. Nakano, and J. Gibson. Block jam: a tangible interface for interactive music. *Journal of New Music Research*, 32(4):383–393, 2003.

[31] J. Noble. Livecoding the synthkit: Little bits as an embodied programming language. In *2014 Second IEEE Working Conference on Software Visualization*, pages 40–44, 2014.

[32] R. Pi. Teach, learn, and make with Raspberry Pi. `https://www.raspberrypi.org/`, 2021. Accessed: 2021-09-01.

[33] V. Potluri, J. Thompson, J. Devine, B. Lee, N. Morsi, P. de Halleux, S. Hodges, and J. Mankoff. Psst: Enabling blind or visually impaired developers to author sonifications of streaming sensor data. In *ACM UIST*, October 2022.

[34] N. Rasamimanana, F. Bevilacqua, N. Schnell, F. Guedy, E. Flety, C. Maestracci, B. Zamborlin, J.-L. Frechin, and U. Petrevski. Modular musical objects towards embodied control of digital music. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction*, pages 9–12, 2010.

[35] J. Sadler, L. Shluzas, P. Blikstein, and R. Katila. Building blocks of the maker movement: Modularity enhances creative confidence during prototyping. *Design thinking research: Making design thinking foundational*, pages 141–154, 2016.

[36] J. D. Sullivan. *Built for Performance: Designing Digital Musical Instruments for Professional Use.* McGill University (Canada), 2021.

[37] S. Tarakajian, D. Zicarelli, and J. Clayton. Mira: Liveness in ipad controllers for max/msp. In *NIME*, pages 421–426, 2013.

[38] G. Wakefield. A streamlined workflow from Max/gen˜ to modular hardware. In *NIME 2021*, apr 29 2021. https://nime.pubpub.org/pub/0u3ruj23.

[39] M. M. Wanderley. Gestural control of music. In *International Workshop Human Supervision and Control in Engineering and Music*, pages 632–644, 2001.

[40] M. Wright, R. J. Cassidy, and M. Zbyszynski. Audio and gesture latency measurements on linux and osx. In *ICMC*, 2004.