

# Towards Interoperable Virtual Reality Musical Instruments in the Musical Metaverse

Alberto Boem  
*Department of Information Engineering  
and Computer Science  
University of Trento  
Trento, Italy  
alberto.boem@unitn.it*

Antony Vitillo  
*R&D Department  
VRROOM Studio  
Ivrea, Italy  
tonyvt@skarredghost.com*

Gad Baruch Hinkis  
*R&D Department  
PatchXR AG  
Volketswil, Switzerland  
gad@patchxr.com*

Lapo Germasi  
*R&D Department  
VRROOM Studio  
Ivrea, Italy  
lapo@vrroom.it*

Luca Turchet  
*Department of Information Engineering  
and Computer Science  
University of Trento  
Trento, Italy  
luca.turchet@unitn.it*

**Abstract**—The emergence of the musical metaverse (MM) raises the problem of interoperability, that is, the ability of different virtual environments to exchange and make use of information, such as transferring virtual objects from one world to another. This is particularly relevant for virtual reality musical instruments (VRMIs), which combine visual form, interaction logic, and real-time sound production, but are typically tightly coupled to a single platform, with no mechanism to carry them across boundaries. While portability has begun to be explored in adjacent domains, these efforts have not extended to the native platforms built on game engines such as Unity that make up much of the current MM. This paper takes a first step toward VRMI portability in this native context. We propose a platform-independent instrument definition model and a serialization-based pipeline that encodes a VRMI’s visual, audio, and interaction properties so that it can be stored, retrieved, and reconstructed elsewhere, together with an adaptation layer that preserves musical behavior while allowing visual form to suit the host platform. As a proof of concept, we transfer two percussive instruments authored in PatchWorld and reconstruct them in Mu-Sync, two Unity-based musical metaverse platforms. Our current implementation shows that one-directional cross-platform VRMI transfer is technically feasible, preserving core audio and interaction properties while adapting visual form to the host platform. While exploratory, the work points to open challenges such as the definition of richer instrument semantics and cross-engine standardization.

**Index Terms**—Musical metaverse, interoperability, virtual reality musical instruments, Unity, XR, social XR

## I. INTRODUCTION

The question of interoperability and platform dependency in networked music systems is not new. Early surveys of network music systems identified how instruments developed for one

system or infrastructure were rarely transferable to another, with performers effectively bound to the specific environment in which their practice was established [1].

Virtual environments (VEs) are increasingly used as a medium for musical creation, performance, and interaction [2]. Within this context, virtual reality musical instruments (VRMIs) [3] represent a particular class of virtual objects that combine visual representation, interaction logic, and real-time sound production, allowing users to perform music in virtual environments [4], [5]. VRMIs may take the form of existing instruments (e.g., [6], [7]) or novel 3D interfaces that exploit the unique affordances of the medium (e.g., [8]–[10]). However, VRMIs have been designed as self-contained artifacts, developed for a single application and therefore tightly coupled to the technical environment in which they were created. This approach reflects the broader trajectory of digital musical instrument design, where instruments are often bespoke, single-use tools rather than portable, reusable artifacts [11].

Unlike conventional digital instruments – which can often be connected through standardized protocols such as MIDI, Open Sound Control (OSC), or plugin formats such as VST – instruments designed for VEs are deeply entangled with the visual and audio rendering engine, interaction system, and runtime architecture of the platform that hosts them. This makes interoperability of VRMIs structurally harder than in earlier networked music systems, and has so far received little explicit research attention.

The emergence of networked VR systems for music has begun to shift this paradigm [12], [13]. Multi-user and collaborative immersive environments now enable shared musical experiences in which performers interact in real time within the same virtual space, and form part of what has been defined as the musical metaverse (MM) [14]. In this context, interoperability—that is, the ability of heterogeneous systems

This work was supported by the MUSMET project funded by the EIC Pathfinder Open scheme of the European Union (grant agreement n. 101184379). Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Innovation Council. Neither the European Union nor the European Innovation Council can be held responsible for them.

to exchange information and make use of it— is not merely desirable but constitutive of the metaverse concept itself. VRMIs should no longer exist as isolated artifacts; instead, they will become components of a shared, persistent musical ecosystem that communities can access across platforms.

Recent technical frameworks converge on the idea that virtual worlds must be traversable and that user identities and virtual assets should persist across platform boundaries [15]–[17]. Without these properties, what emerges is not a metaverse but a collection of isolated silos. In practice, however, only a small number of platforms support the creation of VRMIs (notable examples include PatchWorld by PatchXR<sup>1</sup> and Csound Meta [18]), and instruments created on these platforms cannot be transferred elsewhere without substantial effort, making a continuous and interconnected MM difficult to realize.

Nevertheless, recent work has begun to address similar problems. Buffa et al. [19] proposed using Web Audio Modules (WAM), a browser-native plugin standard built on open W3C specifications, to enable virtual instrument reuse across web-based MM environments. However, many MM platforms are standalone native applications built on game engines such as Unity<sup>2</sup>, and therefore fall outside the scope of such web-standard approaches, leaving the portability problem for native platforms largely unaddressed.

This paper takes a first step toward VRMI portability in the MM by addressing the transfer of instruments between Unity-based platforms through a shared instrument representation and a serialization-based pipeline. We demonstrate this approach with a prototype connecting PatchWorld and Mu-Sync (a custom VR platform for live music events developed by VRROOM Studio<sup>3</sup>) in which VRMIs are serialized, transferred, and reconstructed. Although both platforms use Unity as a common substrate, their different architectures and interaction models make direct asset reuse non-trivial, motivating the use of an intermediate representation.

The prototype is deliberately scoped: it supports one-directional transfer from PatchWorld to Mu-Sync and has been validated on a restricted class of instruments. We therefore frame this work as a feasibility study rather than a claim of full interoperability. The findings show both what a lightweight serialization approach can achieve and the limitations that motivate future work.

This paper makes three contributions: (i) a characterization of the portability gap for VRMIs on native, Unity-based platforms; (ii) a platform-independent definition model for VRMIs capturing visual, audio, and interaction properties decoupled from any single runtime; and (iii) a proof-of-concept pipeline that serializes a VRMI authored in PatchWorld and reconstructs it in Mu-Sync, demonstrating one-directional transfer for a restricted class of instruments.

## II. RELATED WORK

### A. Interoperability in Virtual Environments

Interoperability has been a longstanding challenge in networked VEs. In the context of the metaverse, it specifically refers to the capacity of users to carry their identities, assets, and interactive objects across different platforms without loss of functionality [20]. Despite significant advances, current VEs remain fragmented: different platforms typically operate as closed silos, preventing the seamless movement of content and experience across virtual worlds. Interoperability can therefore be considered at several levels, from the portability of individual assets to the transfer of identity, presence, and interactive behavior.

At the *asset level*, the adoption of open file formats has been the primary strategy for achieving portability. The glTF/GLB format, maintained by the Khronos Group, has emerged as a de facto open standard for 3D model exchange, enabling assets created in one engine or tool to be loaded in another without manual conversion [21].

A particularly relevant extension is *VRM*<sup>4</sup>, a platform-independent format for 3D humanoid avatars [22], [23]. VRM encodes within a single glTF-compliant file all data necessary for cross-platform avatar transfer: standardized bone hierarchy, facial blend shapes, first-person perspective metadata, and embedded license information. These design choices address practical barriers such as differing coordinate conventions and shader pipelines across engines. Within the Japanese ecosystem, VRM has become a widespread standard for avatar portability.

The VRM case is instructive for the problem of VRMI interoperability because it illustrates both the power and the limits of the format-based approach. VRM succeeds precisely because avatars are well-understood artifacts whose structure is sufficiently universal to be captured by a shared schema. VRMIs are considerably more heterogeneous, combining visual geometry, audio behavior, and platform-specific interaction logic – which is why a similarly portable representation for them does not yet exist.

Beyond static assets, interoperability encompasses two further dimensions. The first is *presence and identity*, that is, the ability to carry consistent representations and profiles across platform boundaries, which remains largely unsolved beyond the partial case of VRM [24]. The second, and most relevant for interactive objects, is *behavioral portability*: transferring not only the visual form of an object but also its interaction logic and runtime behavior. OpenXR<sup>5</sup>, for instance, focuses on input abstraction across XR runtimes but does not address object-level behavior. Universal Scene Description<sup>6</sup>

<sup>4</sup>VRM Consortium: 3D humanoid avatar file format for VR, [https://vrm.de/en/vrm/vrm\\_about/](https://vrm.de/en/vrm/vrm_about/) (last accessed: 26/06/2026).

<sup>5</sup>Khronos Group: OpenXR, <https://www.khronos.org/openxr/> (last accessed: 26/06/2026).

<sup>6</sup>Pixar Animation Studios: Universal Scene Description, <https://openscd.org/release/index.html> (last accessed: 26/06/2026).

<sup>1</sup>PatchWorld by PatchXR <https://patchxr.com/> (last accessed: 10/06/2026).

<sup>2</sup>Unity, <https://unity.com/> (last accessed: 10/06/2026).

<sup>3</sup>VRROOM Studio <https://www.vrroom.it/> (last accessed: 14/09/2026).

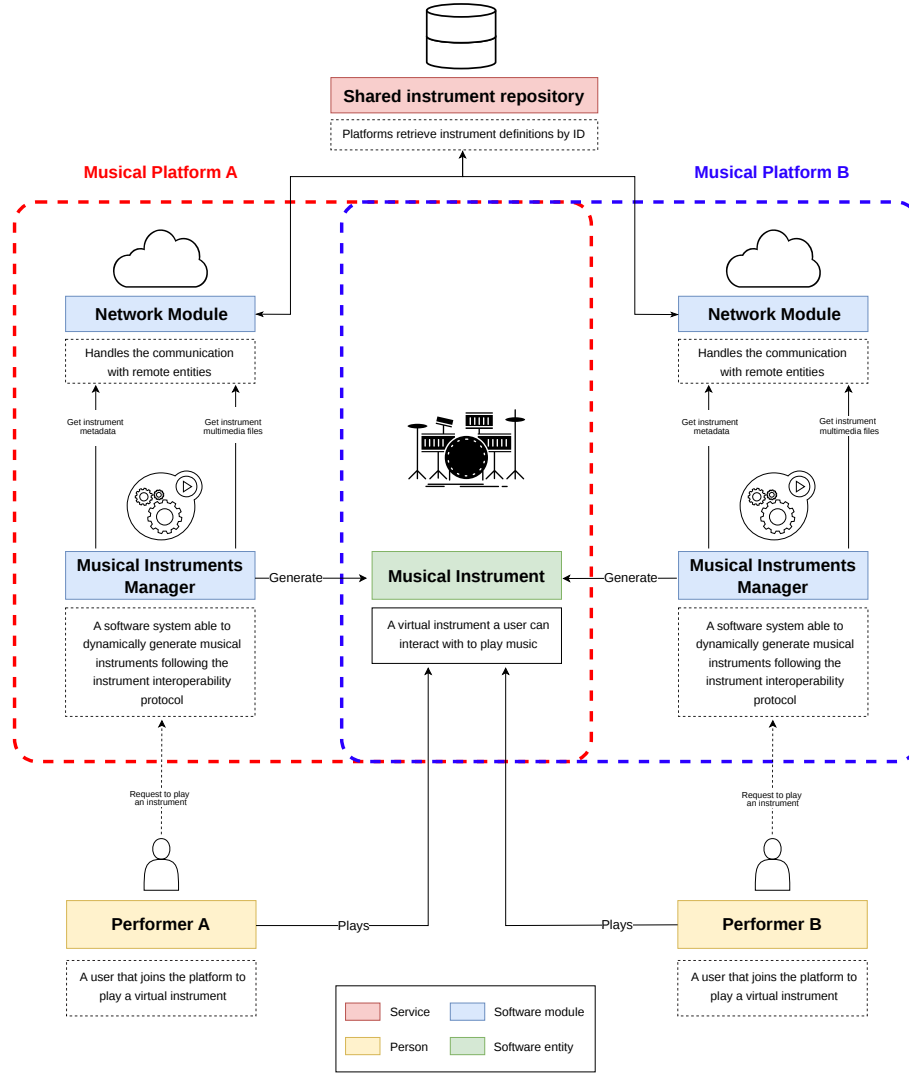


Fig. 1. Conceptual architecture of the instrument interoperability protocol. Each platform exposes a *Musical Instruments Manager*, which reconstructs instruments dynamically according to the protocol, and a *Network Module*, which retrieves instrument metadata and multimedia assets from a shared, platform-external repository. Both platforms can thus instantiate the same instrument, letting performers on different platforms interact with it. The architecture is symmetric and platform-agnostic; the prototype in this paper realizes the one-directional case (PatchWorld to Mu-Sync).

and MPEG-I Scene Description<sup>7</sup> extend scene representation toward interactive semantics, but practical adoption remains limited and neither targets complex interactive musical objects. This gap in representing and transferring objects whose behavior depends on the coupling of gesture, audio, and platform-specific interaction motivates the approach presented in this paper.

### B. Interoperability in the Musical Metaverse

An existing strand of MM research has started to explore web-native approaches, leveraging W3C standards such as the Web Audio API, WebXR, and WebMIDI to build platforms

that run entirely in the browser and are accessible on commercial head-mounted displays. Early prototypes (including the Musical Metaverse Playgrounds [25] and the Orchestra Toolbox [26]) have demonstrated the viability of shared musical experiences in browser-based VEs, but remained confined to a single platform stack and did not address instrument portability across heterogeneous environments.

To date, the most substantial contribution to web-native MM systems has come from Buffa and colleagues, centered on the Web Audio Modules (WAM) standard [27], [28]. WAM defines a browser-native plugin architecture (often described as a web equivalent of VST plugins) that enables modular, reusable audio components to be deployed across different host applications through a standardized API. Building on this,

<sup>7</sup>MPEG-I: Scene Description for MPEG Media, <https://www.mpeg.org/standards/MPEG-I/14/> (last accessed: 26/06/2026).

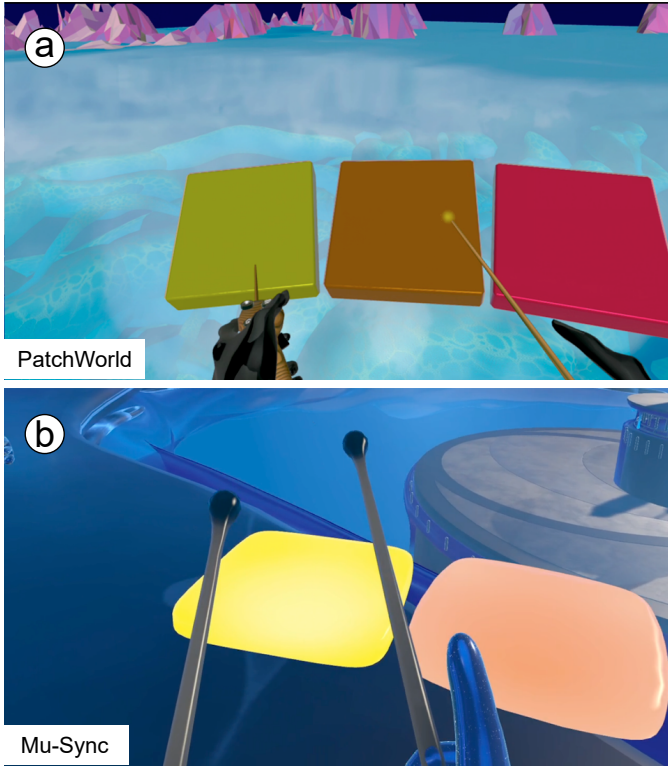


Fig. 2. The same percussive instrument in both platforms: (a) authored and played in PatchWorld, and (b) transferred to and reconstructed in Mu-Sync. The trigger zones and sound are preserved across the transfer, while the visual appearance is adapted to each platform’s rendering style – illustrating functional equivalence without strict uniformity.

Buffa et al. developed *WAM Jam Party* [19], [29], a multi-user immersive environment where participants co-inhabit a shared 3D space and instantiate WAM plugins as spatial objects, including VRMIs that replicate physical instrumental metaphors using physics-based interaction [30]. From an interoperability perspective, WAM’s plugin model provides *component-level portability* within the web ecosystem: the same audio component can be reused across different web-based musical environments without modification.

Our work addresses a complementary dimension: *instrument-level* portability between platforms that are not web-native and do not share the WAM ecosystem. We target Unity-based systems, which represent a significant share of current social and multi-user VR environments and for which no equivalent plugin standard exists. Rather than relying on a shared runtime, our approach defines a serialization format that encodes the essential functional properties of a VRMI (visual assets, audio resources, and interaction parameters) into a platform-independent representation that can be reconstructed in a heterogeneous target environment, prioritizing functional equivalence over strict uniformity.

### III. METHODOLOGY

The primary goal of VRMI interoperability is to enable continuity of musical interaction across platforms. Specifically,

a VRMI used in one platform should: i) be accessible in another, ii) preserve its behavior and interaction logic, iii) provide a consistent musical experience. Thus, the targeted interoperability ensures that a user can retrieve and play the same instrument across platforms while maintaining functional equivalence (Fig. 1). Interoperability here, however, is not limited to carrying an instrument *as-is*: an equally important dimension is its *adaptation* to the host platform. Selected characteristics of a transferred instrument (e.g., its visual form) may be customized to match the aesthetics and conventions of the destination environment. We regard this contextual adaptation as essential to interoperability in the MM (Fig. 2): without it, transferred instruments risk clashing with their surroundings and undermining the coherence of the destination world.

Consider a musician who designs a percussive VRMI (including the interactions, mappings, and sound) and is then invited to perform with it across a series of events hosted on different MM platforms. Despite the effort invested in crafting the instrument, it simply cannot travel: each platform is a closed environment, and the instrument must be rebuilt from scratch each time, with no guarantee of behavioral or sonic consistency across reconstructions.

To address this problem, the methodology adopted in this work consists of three steps: i) designing a shared representation of VR musical instruments, ii) implementing a prototype interoperability pipeline between the two platforms, and iii) demonstrating, as a proof of concept, that instruments created in one platform can be reconstructed and used in the other while preserving their core functionality.

#### A. Platforms

To investigate VRMI portability, we selected two Unity-based MM platforms: PatchWorld and Mu-Sync. First, Unity is a game engine widely used for the development and deployment of VEs. This also includes large-scale social XR applications such as VRChat<sup>8</sup> and Somnium Space<sup>9</sup>, making it a representative substrate for native MM development. Second, the choice of these two platforms reflects practical access: some of the authors of this paper are part of the development teams of the platforms, enabling the level of architectural insight required to implement a cross-platform transfer pipeline. PatchWorld and Mu-Sync embody complementary paradigms. PatchWorld is oriented toward creative authoring, serving as a sandbox for building and playing VRMIs. Mu-Sync is focused on live performance and staging, providing artists with tools to present music to an audience. This asymmetry directly motivates the directionality of the pipeline: transfer from PatchWorld to Mu-Sync reflects a natural workflow in which instruments are first designed in a dedicated authoring environment and then deployed into a performance context. The reverse direction, while technically conceivable, is of lesser practical relevance here and is left for future work.

<sup>8</sup>VRChat <https://hello.vrchat.com/> (last accessed: 10/06/2026).

<sup>9</sup>Somnium Space <https://somniumspace.com/> (last accessed: 10/06/2026).

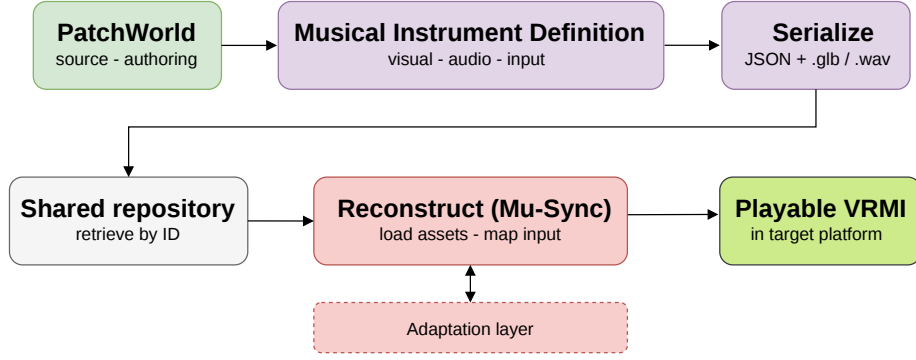


Fig. 3. Transfer pipeline for a VRMI. An instrument authored in PatchWorld is encoded as a platform-independent definition (visual, audio, interaction), serialized to JSON with external asset references (glTF/GLB, WAV), and stored in the shared repository. Mu-Sync retrieves it by identifier and reconstructs it, applying the adaptation layer to override permitted (chiefly visual) properties before the instrument becomes playable.

1) *VRMIs in PatchWorld*: PatchWorld is a social VR platform designed for creating and performing music in VEs, and one of the few platforms that support the design of custom VRMIs directly within VR. In PatchWorld, VRMIs are interactive and spatial entities that combine sound generation, visual representation, and embodied interaction, enabling users to perform music through gestures in 3D space. A key feature is its modular, block-based system, which allows users to assemble custom instruments from components such as sound generators, effects, and interaction modules without requiring traditional programming skills. VRMIs in PatchWorld are inherently spatial and immersive, supporting 3D positioning and real-time interaction, often combined with spatial audio. The platform also integrates with external tools (e.g., MIDI, OSC), allowing instruments to participate in broader music production workflows. In our prototype, PatchWorld serves as the *source* platform: VRMIs authored there are exported into a shared representation and transferred to the target environment.

2) *The Mu-Sync Platform*: Mu-Sync is a platform developed within the MUSMET project<sup>10</sup>, funded by the European Innovation Council [31]. It is designed as a modular social VR environment supporting immersive musical experiences, multiplayer interaction, and cross-platform deployment, integrating networking, spatial audio, avatars, and instrument interaction within a scalable architecture. A defining feature is its cross-platform XR orientation, targeting devices such as Meta Quest, SteamVR, and Pico systems. In our prototype, Mu-Sync serves as the *target* platform: serialized VRMI definitions retrieved from the shared repository are parsed and dynamically reconstructed within its VR environment. Because Mu-Sync was designed with extensibility in mind, it provides a suitable testbed for evaluating whether instruments defined in a heterogeneous source can be instantiated and played with functional equivalence, even in the presence of platform-specific constraints on visual rendering and interaction.

It is worth noting that the two platforms also differ in the networking layer used: PatchWorld relies on Photon PUN 2

for multiplayer synchronization, whereas Mu-Sync uses Mirror together with Unity’s transport layer.

---

#### Algorithm 1 Reconstruct a VRMI in the target platform

---

**Require:** instrument id  $k$ , target platform  $T$

**Ensure:** playable VRMI instance

```

1:  $d \leftarrow \text{Repository.fetch}(k)$  {retrieve serialized definition}
2:  $I \leftarrow \text{parseJSON}(d)$ 
3: {adaptation: override only permitted fields}
4: for all path  $p \in I.\text{adaptable}$  do
5:   if  $T.\text{hasOverride}(p)$  then
6:      $I[p] \leftarrow T.\text{override}(p)$ 
7:   end if
8: end for
9:  $o \leftarrow \text{loadModel}(I.\text{visual.model.uri})$ 
10:  $\text{applyTransform}(o, I.\text{visual.transform})$ ;
     $\text{addToScene}(o)$ 
11:  $a \leftarrow \text{loadSample}(I.\text{audio.sample.uri})$ 
12:  $s \leftarrow \text{createAudioSource}(o, a)$ 
13:  $s.\text{polyphony} \leftarrow I.\text{audio.polyphony}$ ;
     $s.\text{voiceStealing} \leftarrow I.\text{audio.voiceStealing}$ 
14:  $\tau \leftarrow \text{resolveClock}(I.\text{sound.transport}, T)$ 
15:  $c \leftarrow \text{attachCollider}(o, I.\text{interaction.collision.collider})$ 
16:  $\text{bindInteraction}(c, I.\text{interaction}, s, I.\text{sound}, \tau)$ 
17: return  $\text{VRMI}(o, s, I.\text{interaction}, I.\text{sound}, \tau)$ 

```

---



---

#### Algorithm 2 One-shot pad: collision handler

---

**Require:** collision event  $e$ , interaction  $X$ , audio source  $s$ , sound  $P$ , clock  $\tau$

```

1:  $v \leftarrow \text{map}(e.\text{contactSpeed}, X.\text{velocityMapping})$ 
2:  $g \leftarrow P.\text{volume} \cdot v$ 
3:  $\text{trigger}(s, g)$  at  $\text{nextGridPoint}(P.\text{quantization}, \tau)$ 

```

---

## IV. IMPLEMENTATION

The methodology described in the previous section was realized as a prototype pipeline that transfers VRMIs from PatchWorld to Mu-Sync. It comprises three main components:

<sup>10</sup><http://musmet.eu/> (last accessed: 26/06/2026).

---

**Algorithm 3** Sustained instrument: collision handler

---

**Require:** collision event  $e$ , interaction  $X$ , audio source  $s$ , sound  $P$ , clock  $\tau$

```
1: if  $X.retrigger.syncToQuantization$  then
2:    $\Delta \leftarrow \text{gridInterval}(P.quantization, \tau)$ 
3: else
4:    $\Delta \leftarrow X.retrigger.rateMs$ 
5: end if
6: while  $\text{stillInContact}(e)$  do
7:    $\text{trigger}(s, P.volume)$  {restart from sample onset, not looped}
8:   wait  $\Delta$ 
9: end while
```

---

i) a shared instrument representation, ii) a serialization and storage mechanism, and iii) a reconstruction process in the target platform (Fig. 3). We describe each in turn and then introduce the adaptation layer, which mediates platform-specific differences, and the runtime behavior of the reconstructed instruments.

#### A. Instrument Representation

To enable cross-platform interoperability, VRMIs are encoded in a structured, platform-independent definition that captures the properties required to recreate an instrument in a different environment:

- visual components (e.g., 3D models);
- audio assets (e.g., sound samples);
- interaction parameters (e.g., triggering conditions, gesture mappings);
- sound-related parameters (e.g., volume, quantization).

In the current prototype this definition is a structured data object serialized to JSON (Fig. 4). It holds *references* to external multimedia assets – 3D models in glTF/GLB format and audio samples in WAV – rather than embedding them, which keeps the definition lightweight and engine-agnostic. Each definition describes a single sound-producing element: a multi-pad instrument such as the one in Fig. 2 is transferred as a set of definitions sharing a `group` identifier and reconstructed together.

The `audio` block describes engine-level playback (the sample and the allocation of voices), while `sound` holds the musical parameters (level, temporal grid). Both voice allocation and gesture mapping are declared explicitly, since they are precisely the behaviors that drift between engines: `polyphony` and `voiceStealing` fix the number of simultaneous voices and the policy applied when they are exhausted (`oldest`), while `velocityMapping` fixes the source quantity, target parameter, range, and curve relating contact speed to gain.

Since quantization is meaningless without a clock, `sound` also carries a `transport` field: with `source` set to `host` the instrument locks its grid to the destination environment’s clock and the declared `bpm` is ignored, otherwise it runs its own clock at that tempo.

One scope decision concerns audio: in the present prototype, sound is *triggered* (using sample playback) rather than *synthesized* in real time. This reflects a practical constraint of native game engines such as Unity, which lack a portable, high-level real-time audio-graph abstraction, and it marks the main difference from web-native approaches in which audio is generated live within a shared runtime.

The decision to focus on percussive, drum-like instruments is motivated by both practical and musical considerations. Percussive instruments fit naturally within a serialization-based pipeline: their sound production is event-driven, so the audio behavior reduces to a mapping from a discrete interaction condition (a collision, a sustained contact) to a sample, rather than to a continuous synthesis graph that would have to be encoded and faithfully reconstructed across engines.

Moreover, collision- and contact-based interactions are among the most widely and consistently supported input mechanisms across heterogeneous XR platforms, which minimizes the adaptation effort required when an instrument is rebuilt in a different runtime. Musically, percussion is foundational to most musical practice and is particularly well suited to the embodied, gestural nature of VR. In terms of Cadoz’s typology of instrumental gesture [32], [33], the class considered here is restricted to *excitation* gestures. The prototype implements two complementary instruments within the percussive class: a *one-shot* sound pad, in which a strike or collision triggers a single sample, and a *sustained* instrument, in which maintaining contact repeatedly re-triggers a sample (Fig. 5), approximating gestures such as drum rolls or shaken and scraped percussion. Together they support sustained rhythmic interactions while remaining within the trigger paradigm, providing a controlled yet musically meaningful test case.

#### B. Serialization and Storage

Once defined, a VRMI is serialized into the portable format and stored in a shared repository, where it is associated with a unique identifier so that platforms can retrieve it on demand. The serialization captures both the structural and behavioral aspects of the instrument, ensuring its essential characteristics can be reconstructed consistently. The architecture is service-based: a centralized repository acts as an intermediary between platforms, and when a user accesses a VRMI, the host platform queries the repository and retrieves the corresponding serialized definition.

#### C. Reconstruction in the Target Platform

In the target platform (Mu-Sync), the retrieved definition is parsed and then used to dynamically reconstruct the instrument within the VE, as summarized in Algorithm 1. After parsing, the adaptation step (Section IV-D) is applied. The instrument is then rebuilt by loading and placing the 3D assets according to the stored transform, instantiating an audio source bound to the instrument’s sample, attaching a collider, and wiring the interaction logic and sound parameters to the platform’s input system. Reconstruction can occur either at *runtime*, allowing instruments to be loaded dynamically during

execution, or at *design time*, enabling further customization within the development environment. Runtime reconstruction maximizes flexibility, whereas design-time integration allows developers to extend an instrument’s behavior with platform-specific interactions.

#### D. Adaptation Layer

Because platforms differ in their material and shader pipelines, conventions, and interaction mechanics, a faithful byte-for-byte reconstruction is neither always possible nor always desirable. For instance, an instrument authored for PatchWorld may not render or behave well if rebuilt unchanged on another platform. The adaptation layer addresses this by allowing the target platform to override a controlled subset of an instrument’s properties (primarily its visual appearance) while leaving its core behavior intact. The permitted subset is declared explicitly in the definition of the instrument as an adaptable whitelist (e.g., the visual model, its transform, and volume). Fields named in this list may be replaced by target-specific values: all other fields (notably the sample, the triggering condition, the quantization, and the re-trigger rate) are reconstructed verbatim, so the musical behavior of the instrument cannot drift across platforms. As shown in Algorithm 1, adaptation is applied immediately after parsing and *before* any geometry or audio is instantiated, so that the remainder of the reconstruction proceeds identically whether or not an override was supplied. This realizes the principle of *functional equivalence without strict uniformity*: what the instrument does is preserved exactly, while how it looks may bend to the conventions of the host platform.

#### E. Runtime Interaction

Once reconstructed, an instrument can respond to collisions with an *interacting object* (e.g., a virtual mallet, or the user’s hand) that is supplied by the host platform rather than carried in the instrument’s definition. The two instruments, however, differ in how a collision becomes sound. The one-shot pad (Algorithm 2) maps the contact speed of the collision to a gain value, so that harder strikes produce louder output, and triggers a single sample quantized to the musical grid. The sustained instrument (Algorithm 3) does not use contact speed (which carries little meaning for a held, repeated contact) and instead plays at the instrument’s configured volume, re-triggering the sample for as long as contact is maintained at an interval taken from the quantization grid when synchronization is enabled and from an explicit rate otherwise. Each re-trigger restarts the sample from its onset rather than looping a single buffer, producing the rhythmic, roll-like behavior characteristic of sustained percussion.

#### F. Prototype Integration

To validate the implementation, a prototype workflow was developed between PatchWorld and Mu-Sync. The two instruments were authored in PatchWorld, exported into the shared representation, serialized, and imported into Mu-Sync, where they were reconstructed and tested. The prototype

```
{
  "schemaVersion": "0.2",
  "id": "vrmi-pad-kick-001",
  "group": "vrmi-kit-808-001",
  "name": "Kick Pad",
  "class": "percussive",
  "visual": {
    "model": { "uri": "assets/pad_kick.glb", "format": "glb" },
    "transform": {
      "position": [0.0, 1.0, 0.5],
      "rotation": [0.0, 0.0, 0.0],
      "scale": [0.20, 0.05, 0.20]
    }
  },
  "audio": {
    "sample": { "uri": "assets/kick_808.wav", "format": "wav" },
    "polyphony": 4,
    "voiceStealing": "oldest"
  },
  "interaction": {
    "type": "collision",
    "collision": {
      "subtype": "hit",
      "collider": {
        "shape": "box",
        "localPosition": [0.0, 0.0, 0.0],
        "size": [0.20, 0.05, 0.20]
      }
    }
  },
  "velocityMapping": {
    "source": "contactSpeed", "target": "gain",
    "min": 0.0, "max": 1.0, "curve": "linear"
  },
  "sound": {
    "volume": 0.8,
    "quantization": "1/16",
    "transport": { "source": "host", "bpm": 120 }
  },
  "adaptable": [
    "visual.model", "visual.transform", "sound.volume"
  ]
}
```

Fig. 4. Serialized definition of the one-shot percussive pad (collision, hit subtype). A definition describes a single sound-producing element; the multi-pad instrument of Figure 2 is a set of definitions sharing a group identifier. Since `transport.source` is host, the declared bpm is ignored here and applies only if the instrument runs its own clock. The sustained instrument follows the same schema and differs structurally only in its interaction block (Fig. 5).

demonstrates the feasibility of the approach for the percussive class: both the one-shot pad and the sustained instrument were transferred across platforms, preserving their interaction logic and audio output while adapting to Mu-Sync’s visual and interaction context (Fig. 6). We report this as a proof of concept rather than a controlled evaluation, its limitations and the directions they open are discussed in the following section<sup>11</sup>.

## V. DISCUSSION

The prototype presented in this paper shows that a lightweight serialization approach, built around an intermediate instrument representation, is sufficient to move a VRMI across two heterogeneous Unity-based platforms while preserving its interaction logic and audio output. Although the two

<sup>11</sup>A video demonstration of the system is available at: <https://youtu.be/JVYweMs-bCM> (last accessed: 30/06/2026).

```

"interaction": {
  "type": "collision",
  "collision": {
    "subtype": "penetration",
    "collider": {
      "shape": "capsule",
      "localPosition": [0.0, 0.0, 0.0],
      "size": [0.10, 0.30, 0.10]
    }
  },
  "retrigger": {
    "whileInContact": true,
    "playback": "restart",
    "rateMs": 120,
    "syncToQuantization": true
  }
}

```

Fig. 5. The `interaction` block of the sustained instrument. While contact is maintained, the sample is re-triggered (restarted from its onset) rather than looped as a single sustained buffer. Because `syncToQuantization` is true, the re-trigger interval follows the quantization grid of the host clock (e.g., 125 ms for a 1/16 note at 120 bpm); `rateMs` is used only when synchronization is disabled. No `velocityMapping` is declared: contact speed carries little meaning for a held contact, so the instrument plays at its configured volume.

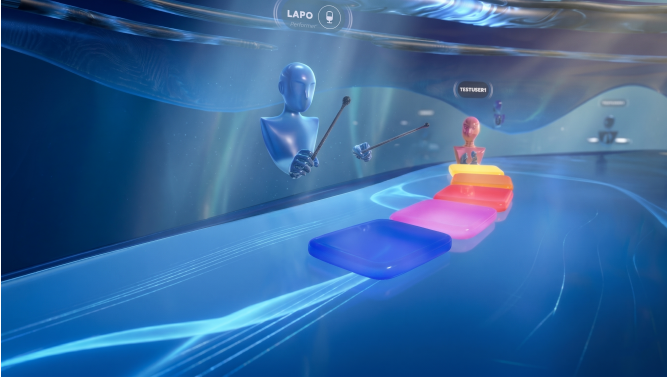


Fig. 6. The reconstructed percussive instrument in use within Mu-Sync. A performer plays the transferred sound pads with platform-supplied mallets; the pads retain the trigger behavior and audio of the original while their appearance follows Mu-Sync’s visual style.

platforms share a common engine, they differ substantially in instrument architecture and interaction model. The significance of this result lies less in the specific pipeline than in what it implies for the broader MM: if instruments — not only avatars or static assets — can be described declaratively and reconstructed elsewhere, then platforms need not remain isolated silos, and a performer’s personal setup can begin to travel with them across virtual worlds. This complements rather than competes with web-native approaches such as WAM [19]: whereas those achieve component-level portability within a shared browser runtime, our approach targets portability *between* native runtimes that share no such substrate.

Although our prototype connects two Unity-based platforms, nothing in the representation presupposes a particular engine: it is built on open, engine-neutral formats (glTF/GLB for geometry, WAV for audio, JSON for structure), so the same definitions could in principle be consumed by Unreal, web, or mobile runtimes. In this sense the approach targets a genuinely heterogeneous MM rather than a single engine’s ecosystem.

Therefore, the aim is not a binary plugin interface for game engines — a “VST for Unity” — but a declarative instrument description, in the lineage of glTF and its extensions, from which each runtime builds its own native objects.

A key lesson is that the quality of serialization depends directly on how well the underlying instrument can be described. Our representation, however, is tailored to percussive, trigger-based instruments. Extending it to the full diversity of VRMIs (e.g., continuous controllers, synthesis-driven instruments, spatially distributed interfaces) is not merely a matter of adding fields, but of agreeing on *what* should be described in the first place. This points to the need for a dedicated taxonomy, or descriptive dimension space, for VRMIs: a principled vocabulary characterizing instruments along the axes most relevant to transfer, such as the type of gesture and interaction, the sound-production model (sample playback versus real-time synthesis), the gesture-to-sound mapping, and the spatial and visual form of the instrument. Gestural typologies [33] and dimension spaces for digital musical instruments [34], [35] offer a useful starting point (the excitation/modification/selection distinction already separates what the present schema can carry from what it cannot), however they were not conceived for the compound visual, spatial, and audio nature of VRMIs [3]. Such a framework would let a serialization schema be derived systematically from instrument categories rather than reverse-engineered case by case, and we regard its development as a precondition for scaling the approach beyond the restricted class considered here.

## VI. LIMITATIONS

Several limitations bound our claims. The pipeline is *one-directional*, flowing from PatchWorld (authoring) to Mu-Sync (performance) in line with a natural creative workflow, but stopping short of bidirectional interoperability. It has been validated on a *single, restricted class of instruments* (percussive, trigger-based instruments), and the current definition does not yet capture richer features such as continuous control, animation, visual effects, or complex interaction behaviors, which limits how faithfully more elaborate VRMIs can be reproduced. Most consequentially, audio is *triggered sample playback rather than real-time synthesis*. This is a deliberate shortcut that distinguishes our approach from component-level web solutions, where audio is generated live within a shared runtime [30], and that partly reflects the absence of a native, portable real-time audio-graph abstraction in Unity, where low-level DSP is accessible only through callbacks such as `OnAudioFilterRead`.

At the *asset level*, relying on standard formats such as GLB ensures a degree of compatibility but constrains the supported feature set, and differences in platform-specific rendering pipelines can yield visual variation even when functional equivalence is preserved.

A subtler limitation concerns where the boundary of the transferred artifact is drawn. The definition describes the instrument itself (its body, trigger zones, and sound) but deliberately omits the *interacting object* (e.g., a mallet, drumstick,

or the user’s hand) that acts upon it. That object is provided by the host platform and is the entity that must be network-synchronized; delegating it to the platform keeps the approach compatible with each platform’s own networking stack (PatchWorld’s Photon PUN 2, Mu-Sync’s Mirror) without having to reconcile them. This separation works cleanly for percussion, where the striking implement is generic and platform-supplied, but it does not generalize: many instruments cannot be split into a transferable “struck” part and a platform-supplied “striker,” so extending the approach to them remains an open problem.

Relatedly, the `transport` field resolves the clock a single instrument locks to, but not the alignment of clocks across hosts: sustaining a shared grid between a PatchWorld and a Mu-Sync instance would require reconciling Photon PUN 2 and Mirror timing, which the current prototype does not attempt.

Finally, in the current implementation the adaptation of the instrument between platforms is performed manually. Future implementations might involve a layer of automation and AI-assisted adaptivity.

## VII. FUTURE WORK

The limitations described above define a clear agenda. Defining a set of richer instrument semantics would extend the instrument representation beyond sample triggers, potentially by cross-compiling DSP descriptions into engine-native audio callbacks. This also bears on the *runtime-versus-design-time* trade-off, since deeper environmental integration currently comes at the cost of portability. Bidirectional and multi-platform transfer would require a round-trip-safe representation and the reconciliation of divergent synchronization models to support networked, multi-user performance. Standardization could replace the bespoke format with a declarative “VRM-for-instruments” schema, following the trajectory by which VRM is being proposed as an official glTF extension and aligning with broader scene-level standards. Finally, a formal evaluation with musicians is needed to assess whether transferred instruments preserve not only function but also the perceived quality and expressivity of the original, across a wider range of instrument types and target platforms.

## VIII. CONCLUSION

This paper addressed the portability of VRMIs across heterogeneous platforms dedicated to the musical metaverse, where fragmentation forces musicians to recreate their instruments whenever they move between environments. We proposed a serialization-based approach in which a VRMI is encoded in a portable, platform-independent representation, stored in a shared repository, and dynamically reconstructed in a target platform. As a proof of concept, we transferred two percussive instruments from PatchWorld to Mu-Sync, showing that an instrument’s core interaction logic and audio behavior can be preserved across platforms even under platform-specific constraints.

Beyond the technical prototype, the work points to a broader principle: interoperability in the musical metaverse should aim to preserve functionality and user experience rather than enforce strict uniformity, striking a balance between portability and platform-specific adaptation that diverse creative ecosystems require. The approach remains limited in scope – one-directional, restricted to a single instrument class, and not yet evaluated with users – and realizing fully portable setups will ultimately depend on richer instrument representations and shared standards. Even so, it marks a concrete step toward a metaverse in which musicians can carry their instruments, performances, and creative identities across different virtual worlds.

## REFERENCES

- [1] Á. Barbosa, “Displaced soundscapes: A survey of network systems for music and sonic art creation,” *Leonardo Music Journal*, vol. 13, pp. 53–59, 2003.
- [2] L. Turchet, R. Hamilton, and A. Çamcı, “Music in extended realities,” *IEEE Access*, vol. 9, pp. 15 810–15 832, 2021.
- [3] S. Serafin, C. Erkut, J. Kojs, N. C. Nilsson, and R. Nordahl, “Virtual reality musical instruments: State of the art, design principles, and future directions,” *Computer Music Journal*, vol. 40, no. 3, pp. 22–40, 2016.
- [4] T. Mäki-Patola, J. Laitinen, A. Kanerva, and T. Takala, “Experiments with virtual reality instruments,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2005, pp. 11–16.
- [5] F. Berthaut, “3D interaction techniques for musical expression,” *Journal of New Music Research*, vol. 49, no. 1, pp. 60–72, 2020.
- [6] I. Hwang, H. Son, and J. R. Kim, “AirPiano: Enhancing music playing experience in virtual reality with mid-air haptic feedback,” in *2017 IEEE World Haptics Conference (WHC)*. IEEE, 2017, pp. 213–218.
- [7] D. Johnson and G. Tzanetakis, “VRMin: using mixed reality to augment the theremin for musical tutoring,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2017, pp. 151–156.
- [8] V. Zappi, D. Mazzanti, A. Brogni, and D. Caldwell, “Concatenative synthesis unit navigation and dynamic rearrangement in VRGrains,” in *Proceedings of the sound & music computing conference*, 2012, pp. 403–408.
- [9] R. Hamilton and C. Platz, “Coretet: A dynamic virtual musical instrument for the twenty-first century,” in *2019 IEEE Conference on Virtual Reality and 3D User Interfaces (VR)*. IEEE, 2019, pp. 1510–1512.
- [10] A. Çamcı, M. Vilaplana, and R. Wang, “Exploring the affordances of VR for musical interaction design with VIMes,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2020, pp. 121–126.
- [11] F. Morreale and A. McPherson, “Design for Longevity: Ongoing Use of Instruments from NIME 2010-14,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*. Copenhagen, Denmark: Aalborg University Copenhagen, 2017, pp. 192–197.
- [12] B. Loveridge *et al.*, “Networked music performance in virtual reality,” *Journal of the Audio Engineering Society*, 2020, workshop paper.
- [13] A. Boem, M. Tomasetti, L. Turchet *et al.*, “Harmonizing the musical metaverse: unveiling needs, tools, and challenges from experts’ point of view,” in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2024, pp. 206–214.
- [14] L. Turchet, “Musical metaverse: Vision, opportunities, and challenges,” *Personal and Ubiquitous Computing*, vol. 27, no. 5, pp. 1811–1827, 2023.
- [15] J. D. N. Dionisio, W. G. B. III, and R. Gilbert, “3D virtual worlds and the metaverse: Current status and future possibilities,” *ACM Computing Surveys (CSUR)*, vol. 45, no. 3, pp. 1–38, 2013.
- [16] C. Perey, “Interoperability is a fundamental requirement for the open metaverse,” in *2024 IEEE International Symposium on Emerging Metaverse (ISEMV)*. IEEE, 2024, pp. 21–24.
- [17] L. Yang, S.-T. Ni, Y. Wang, A. Yu, J.-A. Lee, and P. Hui, “Interoperability of the metaverse: A digital ecosystem perspective review,” *IEEE Engineering Management Review*, 2025.

- [18] H. Vo and R. Boulanger, "Csound in the MetaVerse – From Cabbage to CsoundUnity and Beyond: Developing a Working Environment for SoundScapes, SoundCollages, and Collaborative SoundPlay," in *Proceedings of the 7th International Csound Conference*. Vienna, Austria: ICSC 2024, September 2024, pp. 115–119.
- [19] M. Buffa, D. Girard, and A. Hofr, "Using Web Audio Modules for Immersive Audio Collaboration in the Musical Metaverse," in *2024 IEEE 5th International Symposium on the Internet of Sounds (IS2)*, 2024, pp. 1–10.
- [20] Metaverse Standards Forum, "Metaverse standards forum: Overview and working groups," Metaverse Standards Forum, Tech. Rep., 2022, accessed: 2026. [Online]. Available: <https://metaverse-standards.org>
- [21] The Khronos Group, "glTF 2.0: GL transmission format specification," The Khronos Group, Tech. Rep., 2017, accessed: 2026. [Online]. Available: <https://www.khronos.org/glTF/>
- [22] VRM Consortium, "VRM – extending glTF for portable humanoid avatars," Presentation at SIGGRAPH Asia 2019, Brisbane, Australia, Nov. 2019. [Online]. Available: [https://www.khronos.org/assets/uploads/developers/presentations/VRM-Extending-glTF-for-Portable-Humanoid-Avatars\\_SIGGRAPH-Asia\\_Nov19.pdf](https://www.khronos.org/assets/uploads/developers/presentations/VRM-Extending-glTF-for-Portable-Humanoid-Avatars_SIGGRAPH-Asia_Nov19.pdf)
- [23] —, "VRM: A 3D avatar file format for the VR era — technical specification," VRM Consortium, Tech. Rep., 2018, version 0.x; accessed: 2026. [Online]. Available: <https://vrm.dev/en/>
- [24] S. Mystakidis, "Metaverse," *Encyclopedia*, vol. 2, no. 1, pp. 486–497, 2022.
- [25] A. Boem and L. Turchet, "Musical Metaverse Playgrounds: Exploring the Design of Shared Virtual Sonic Experiences on Web Browsers," in *2023 4th International Symposium on the Internet of Sounds (IS2)*. IEEE, 2023.
- [26] D. Dziwis, H. von Coler, and C. Pörschmann, "Orchestra: A toolbox for live music performances in a web-based metaverse," *Journal of the Audio Engineering Society*, vol. 71, no. 11, pp. 802–812, 2023.
- [27] J. Kleimola and O. Larkin, "Web Audio Modules," in *Proceedings of the 12th Sound and Music Computing Conference (SMC)*. Maynooth, Ireland: Maynooth University, 2015, pp. 249–256.
- [28] M. Buffa, S. Ren, O. Campbell, J. Kleimola, O. Larkin, and T. Burns, "Web Audio Modules 2.0: An open Web Audio plugin standard," in *Companion Proceedings of the ACM Web Conference 2022 (WWW '22 Companion)*. Lyon, France: ACM, 2022, pp. 364–369.
- [29] M. Buffa, M. Winckler, Q. Escobar, S. Demont, A. Hofr, and A. Mir-Sadjadi, "Creating collaborative and immersive modular audio installations with WAM Jam Party," in *IS2 2025 – IEEE International Symposium on the Internet of Sounds*. L'Aquila, Italy: IEEE, Oct. 2025.
- [30] M. Buffa, M. Winckler, and A. Mir-Sadjadi, "Embodied virtual instruments in web-based multi-user VR: A case study with a 3D drum kit and Web Audio Modules," in *Proceedings of the 9th Web Audio Conference (WAC 2025)*, Paris, France, Nov. 2025.
- [31] L. Turchet *et al.*, "Towards the musical metaverse: The MUSMET project," in *Proceedings of the IEEE International Symposium on the Internet of Sounds (IS2)*. IEEE, 2025.
- [32] C. Cadoz, "Instrumental gesture and musical composition," in *Proceedings of the International Computer Music Conference (ICMC)*, Cologne, Germany, Feb. 1988, pp. 1–12.
- [33] C. Cadoz and M. M. Wanderley, "Gesture-music," *Trends in Gestural Control of Music*, 2000.
- [34] D. Birnbaum, R. Fiebrink, J. Malloch, and M. M. Wanderley, "Towards a dimension space for musical devices," in *Proceedings of the International Conference on New Interfaces for Musical Expression (NIME)*, Vancouver, BC, Canada, 2005, pp. 192–195.
- [35] S. O'Modhrain, "A framework for the evaluation of digital musical instruments," *Computer Music Journal*, vol. 35, no. 1, pp. 28–42, 2011.