

A Description-Driven XR Translation of VCV Rack Modular Synthesis Interfaces

Giuseppe Bergamino, Michael Fioretti, Leonardo Gabrielli, and Stefano Squartini

Department of Information Engineering

Università Politecnica delle Marche

Ancona, Italy

{g.bergamino, m.fioretti}@pm.univpm.it; {l.gabrielli, s.squartini}@staff.univpm.it

Abstract—Extended reality (XR) enables musical interfaces to be reimagined as spatial, embodied, and networked interaction environments. Current approaches include self-contained XR musical systems, external software connected through MIDI/OSC mappings, and architectures that integrate musical software components directly into immersive runtimes. This paper explores a complementary integration model in which the source software environment remains active as the computational core, while XR acts as a synchronized spatial client derived from its interface structure. We present VeryVCV Rack, a prototype that applies this model to VCV Rack by reconstructing a mixed reality representation of the live patch from structured interface information. Description-driven translation uses machine-readable information about modules, controls, layout, parameter values, interaction roles, and state to share the interaction knowledge needed to reconstruct and synchronize the spatial interface. A preliminary system characterization examines description scalability, interface-description coverage, and runtime synchronization behavior, identifying the feasibility and current implementation constraints. The prototype is released to support reproducibility and future extensions.

Index Terms—Musical Metaverse, Extended Reality, Mixed Reality, Modular Synthesis, VCV Rack.

I. INTRODUCTION

Digital music-making increasingly involves software instruments, networked devices, and mediated interaction environments. Extended reality (XR) offers ways to move musical interfaces beyond two-dimensional screens through spatial interaction, embodied control, and networked participation [1]. These developments contribute to the broader vision of the Musical Metaverse (MM), framed as a convergence of Musical XR, networked music performance, and Internet of Musical Things (IoMusT) technologies [2]. However, current MM systems still face technical and design challenges, particularly for music creation and performance [3].

Modular synthesis provides a relevant musician-centered domain for XR because interaction is inherently spatial, exploratory, and embodied. Bodily engagement in musical interaction has been addressed through embodied music cognition and accounts of how instrument structure and material properties participate in musical action and thinking [4], [5]. Hardware modular synthesizers make these relationships tangible through modules, controls, jacks, and patch cables [6], whereas software environments such as Pure Data, Max/MSP, and VCV Rack retain modular patching while largely relocat-

ing interaction to two-dimensional, pointer-based interfaces. XR research has explored alternative spatial realizations of modular synthesis and patching [7], [8].

Many XR musical systems are designed as self-contained immersive instruments, allowing interaction, representation, and audio processing to be developed together but binding them to a specific XR application. Other approaches connect existing devices or software through protocols such as MIDI and OSC, typically exchanging control data without exposing the structure and functional relationships of the controlled interface [9], [10]. WAM Jam and PdXR go further by using structured software information to construct immersive representations of existing musical components or patches [11]–[13]. This paper addresses a complementary integration problem: adding XR interaction to a running desktop music environment while retaining its processing and established workflow. Inspired by interface-semantic models such as WAI-ARIA [14], the proposed description-driven model exposes machine-readable information about interface elements, their parameters, layout, roles, states, and relationships. The XR client uses this interaction knowledge to reconstruct and synchronize a spatial interface, following an integration rather than migration model.

We introduce VeryVCV Rack as a prototype instantiation of this integration model. It applies description-driven translation to VCV Rack, a software environment for developing and performing with virtual modular synthesizers [15]. As illustrated in Figure 1, VCV Rack remains responsible for the musical patch, audio computation, and parameter state, while a dedicated module exposes a structured description of the live patch. A Meta Quest 3 client uses this description to reconstruct panels and controls in MR and maintain bidirectional parameter synchronization.

The contribution of this paper is threefold. First, it formulates a description-driven XR integration model in which a running music-software environment exposes structured interaction information to a synchronized spatial client (Section III). Second, it presents VeryVCV Rack as an implementation of this model for VCV Rack and MR (Section IV). Third, it reports a preliminary system characterization addressing description scalability, interface-description coverage, and observable synchronization behavior, without assessing usability or musical suitability (Section V). The paper concludes by



Fig. 1. Overview of the VeryVCV Rack case study. Left: a patch running in VCV Rack on the desktop computer. Right: the corresponding reconstruction of the patch as seen from the Meta Quest 3 headset.

discussing the trade-offs of the proposed model and its implications for integrating established music-software ecosystems with emerging Musical Metaverse workflows (Section VI).

II. BACKGROUND

A. Modular Synthesis in Software

Over the last decade, modular synthesis has attracted renewed interest alongside the diffusion of the Eurorack format and has been discussed in relation to materiality, tactility, constraint-based creativity, and the social practices surrounding modular systems [6], [16]. Software modular environments preserve the functional logic of modular patching while reducing constraints related to physical space and module availability and enabling larger, recallable, and shareable configurations. Among them, VCV Rack explicitly models the Eurorack paradigm in software [17]. At the time of writing, the VCV Library lists more than 4100 modules from over 300 brands and developers, most of them open-source [18]. Its C++ SDK and plugin API [19] combine sample-by-sample DSP processing with graphical components for module panels and user interaction [15].

Hybrid physical interaction with VCV Rack has also been explored in Umbilical, which couples the software environment with a customizable modular MIDI controller [20]. VCV Rack’s API and heterogeneous module ecosystem make it a suitable source environment for investigating description-driven XR translation across diverse module interfaces.

B. Modular Synthesis in XR

Several works have explored how the spatial, tactile, and exploratory qualities of modular synthesis can be reinterpreted in XR. Wakefield et al. investigated modular synthesis in VR, focusing on patching, control manipulation, and spatial organization beyond physical racks and two-dimensional software interfaces [7]. Palumbo et al. similarly examined modular patching as a paradigm for three-dimensional interaction and spatialized representations of signal flow [21].

A related line of work has developed XR-native modular environments. OpenSoundLab provides a mixed-reality sound laboratory supporting spatial patching, embodied interaction, and walkable modular instruments [8], [22]. More recent work compares OpenSoundLab and PatchWorld as examples of emerging XR modular practices and situates platforms such as SynthVR, SYNTHSPACE, MuX, and Virtuoso within the broader landscape of virtual modular instruments [16]. These systems establish XR as a native medium for modular synthesis, where instrument representation, interaction, and sound processing can be designed together within the immersive environment. At the same time, XR modular systems introduce their own material and infrastructural requirements, including head-mounted hardware and dependence on specific software and platform ecosystems [16]. This paper builds on this literature while shifting the focus from designing a new XR modular instrument to translating an existing software environment into a spatial MR interaction layer.

C. Interoperability and Networked Musical XR

Beyond XR-native modular instruments, a related research direction concerns how immersive musical environments interoperate with existing devices, software tools, and networked infrastructures. This issue is central to IoMusT and Musical Metaverse research, where musical interaction is distributed across heterogeneous instruments, applications, performers, and environments [2], [23]. A common strategy is to connect external devices or software through device-agnostic protocols such as MIDI and OSC. OSC-XR provides a Unity-based toolkit for immersive musical interfaces communicating through OSC [9], while MUSMET-Bridge translates interaction with USB MIDI control surfaces into OSC messages for standalone VR headsets [10]. Such approaches support heterogeneous tools, but primarily exchange control data rather than descriptions of the structure and functional relationships of the controlled interface.

A more application-specific connection between modular music software and real-time graphics is provided by URack, which connects VCV Rack to Unity through dedicated modules that use Rack parameters to drive graphical objects and visual properties in a Unity scene, while audio generation and processing remain within VCV Rack [24]. Closer to the present work, WAM Jam and PdXR show how descriptions and structural information from established musical software can be used to construct immersive interfaces. WAM Jam hosts Web Audio Modules as interoperable audio components within a WebXR environment and generates 3D interfaces from parameter descriptions, with additional mappings supporting customized spatial representations [11], [12]. PdXR similarly enables patches created through the established Pure Data workflow to run within web-based XR environments, where supported GUI elements are translated into interactive three-dimensional components [13].

This paper explores a different relationship between the musical environment and XR. Rather than making the immersive application the host of the musical component or runtime, the complete application on the desktop remains concurrently active and responsible for audio processing, patch editing, state, and hardware integration, while the XR application acts as a synchronized client of this running environment. The proposed model thus combines continuity of the native music-software environment with description-based spatial reconstruction: the client receives not only parameter updates but structured information about controls, roles, states, layout, and their relationships. In this sense, integration rather than migration concerns both where musical computation remains and what is shared with XR, extending interoperability beyond control-message exchange toward the interaction knowledge required to reconstruct and synchronize an alternative spatial interface.

III. DESCRIPTION-DRIVEN XR INTEGRATION MODEL

The proposed model takes as a conceptual starting point the semantic interface representations used in accessibility architectures, where applications expose structured functional descriptions of interface elements to external clients. In this context, interface semantics denotes the information that makes an interface element identifiable as an actionable component rather than as a purely graphical object, specifying what it represents, how it can be operated, and how it relates to the surrounding interface structure [14], [25].

The Web Accessibility Initiative–Accessible Rich Internet Applications (WAI-ARIA) specification, for example, defines roles, states, and properties that allow interactive web components to expose their semantics to accessibility clients and assistive technologies [14]. Similarly, audio software frameworks such as JUCE provide mechanisms for exposing functional information about interface components to platform accessibility clients [26]. Outside accessibility, the Document Object Model (DOM) provides a related paradigm by representing web documents as structured objects that can be accessed and manipulated independently of their rendered appearance [27].

These examples share the principle that an interface can be exposed as a structured operational description rather than only as a visual surface. Here, this principle is applied to musical XR: a source music environment exposes interaction-relevant information about its interface, while an XR client uses that information to construct and synchronize an alternative spatial interaction surface.

A. Design Principles

The model is characterized by four design principles. They describe the integration strategy proposed in this paper, while Section IV instantiates them in the specific case of VCV Rack.

Musical-software continuity: the source music-software environment remains operational as the computational and workflow core, while XR is introduced as an additional interaction surface. Audio processing, parameter application, desktop operation, and integration with external musical hardware therefore remain anchored to the source environment.

Source-native interface description: the XR representation is derived from structured information exposed by the source environment, or by an extension of it, rather than from visual analysis of its screen-based representation. The description captures interaction-relevant information about interface elements, including their identifiers, labels, roles, states, values, and relationships.

Spatial affordance translation: the XR client maps described interface elements to spatial interaction components according to their roles and properties. The resulting representation may preserve selected correspondences with the source interface, such as naming, grouping, and layout relationships, to support continuity and orientation between desktop and XR interaction.

Bidirectional state consistency: interactions performed through either the source environment or the XR client remain synchronized. The source environment retains authority over parameter values and state changes: control data produced through XR interaction are applied by the source environment, while resulting state changes are propagated back to the XR representation.

B. Model Components

Figure 2 summarizes the components and information flows of the model. On the computer side, the *Source Environment* is the running music-software system that hosts the original instrument or patch and exposes, directly or through an extension, the information required by the XR client. This information is represented by the *Interface Model*, which describes interface elements through identifiers, labels, roles, states, values, and spatial or hierarchical relationships.

The *Connection Layer* mediates communication between the source environment and the XR client through two distinct information flows. The *Description* flow communicates the interaction knowledge required to interpret and reconstruct the source interface, including its elements and their functional and spatial relationships. The *Updates* flow carries the control and state changes that occur once this structure has been

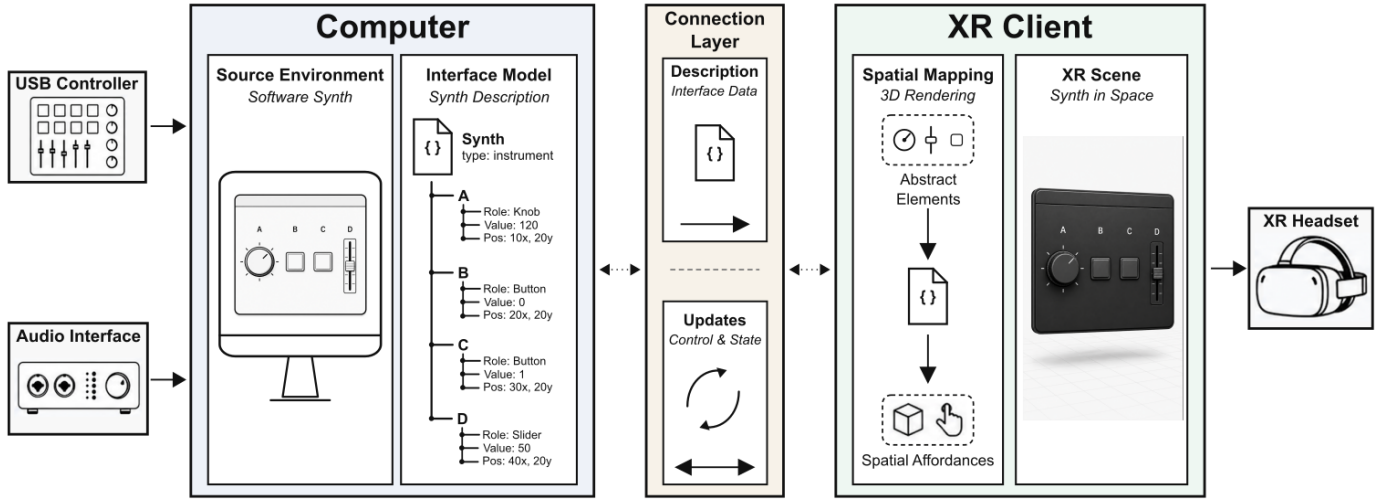


Fig. 2. Description-driven XR integration model. A source music environment exposes a structured interface model; a connection layer separates interface description from runtime control and state updates; and an XR client translates the received information into spatial affordances.

shared. It is bidirectional: actions performed in XR are sent to the source environment and vice versa. This separation distinguishes information describing *what* the interaction structure is from messages describing *what changes* during use.

On the XR side, the *Spatial Mapping* stage interprets the received description and uses interface roles and properties to determine corresponding spatial interaction components or behaviors, such as knobs, buttons, or sliders. The *XR Scene* places these components in a spatial context and maintains their relationship with the corresponding source elements, producing an alternative interaction surface that remains synchronized with the running music-software environment.

IV. CASE STUDY: VERYVCV RACK

VeryVCV Rack instantiates the proposed integration model with VCV Rack as the Source Environment and a standalone Meta Quest 3 application as the MR client. VCV Rack is a suitable case study because its documented C++ plugin API exposes Rack-native abstractions for modules, graphical components, and parameters [28]. In parallel, the VCV Library provides a large and active ecosystem of modules with heterogeneous interface designs, developed across different contributors and use cases [18]. The source code for both the VCV Rack-side Bridge module and the Unity-based MR client is provided in the companion repository [29].

Although the integration model is not specific to a particular XR modality, VeryVCV Rack targets mixed reality. MR allows performers to see and use the surrounding physical environment while interacting with virtual elements, supporting hybrid setups that combine spatial controls with existing musical hardware and external controllers. Consistent with this use case, the prototype relies on hand tracking rather than handheld XR controllers, allowing performers to move between physical instruments and spatial controls without holding dedicated interaction devices. Figure 3 illustrates the intended use scenario.

The MR client is implemented in Unity for the Quest 3 headset, following Meta’s official Quest development workflow [30]. VCV Rack runs on a desktop computer and hosts a custom Bridge module that generates a structured patch description, exposes it to the client, receives control messages produced through MR interaction, and applies them to the corresponding Rack parameters.

The current implementation focuses on interactive reconstruction of standard VCV Rack controls, namely knobs, buttons, switches, and sliders. Jacks and labels are also represented as part of the reconstructed interface, but jacks are currently non-interactive because cable editing and routing are outside the scope of the prototype.

A. VeryVCV Bridge Module

The *VeryVCV Bridge* is implemented as a custom module inserted into a VCV Rack patch. It exposes a structured description of the live patch interface and mediates runtime parameter synchronization with the MR client.

The Bridge generates this representation through an explicit scan operation, performed directly on VCV Rack’s internal patch structure via its API. When triggered by the user, the scan builds an inventory of the modules currently present in the patch. For each module, it records the information needed to identify it and preserve its position and size within the Rack layout. It also collects the controls and jacks associated with each module, including their unique identifiers and local position and size on the panel. These data allow the MR client to reconstruct a layout coherent with the original VCV Rack patch and to bind each interactive spatial control to the corresponding VCV Rack parameter.

For each controllable parameter, the Bridge additionally records labels, roles, interaction behaviors, and value information. Roles identify common interface elements such as knobs, sliders, buttons, and switches, while interaction behaviors distinguish continuous, stepped, momentary, and toggle

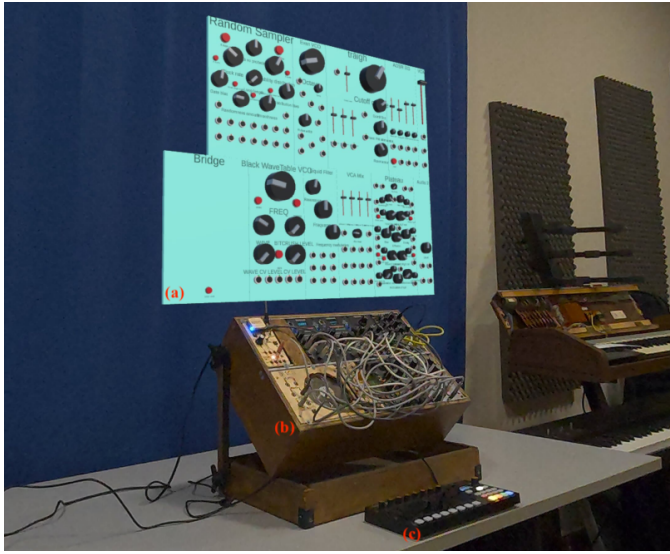


Fig. 3. MR view of VeryVCV Rack on the Meta Quest 3, illustrating a possible hybrid performance setup in which spatially reconstructed VCV Rack controls (a) coexist with physical modular hardware (b) and a USB MIDI controller (c).

interactions. This allows the MR client to select appropriate spatial components without module-specific mappings for individual controls. The resulting description is stored as a *patch snapshot*, while runtime parameter changes are synchronized bidirectionally between VCV Rack and the MR client, with VCV Rack retaining authority over parameter state.

B. Structured Patch Snapshot

In VeryVCV Rack, the *Interface Model* defined in Section III-B is instantiated by the *patch snapshot* generated by the Bridge module. The snapshot follows a JSON-based operational schema for exchanging patch and interface information. JSON was selected because it is directly machine-readable by the system components while remaining human-inspectable during development, debugging, and documentation. The schema is not intended as a universal ontology of VCV Rack or modular synthesis; rather, it defines an operational subset of information required by the current system to reconstruct interface elements in MR and maintain parameter-level synchronization.

The JSON snapshot can be read as a four-level functional description of the Rack patch:

- 1) *Patch-level information*: metadata, layout information, and the bounds of the scanned Rack area.
- 2) *Module-level information*: one entry for each scanned module, including its source identity, name, size, and position within the Rack layout.
- 3) *Interface-element information*: controls and jacks associated with each module, including their identifiers, roles, and position and size on the module panel; controllable elements additionally include labels and parameter associations.

- 4) *Interaction-level information*: data associated with controllable elements, including behavior type, value range, current and normalized values, unit, and display string.

This hierarchical organization allows the MR client to reconstruct the patch progressively: it places modules according to the exported Rack layout, instantiates their spatial interface elements, and binds interactive controls to the corresponding Rack parameters for bidirectional synchronization. Figure 4 shows a selected structural excerpt from a generated patch snapshot. The excerpt preserves the overall hierarchy of the description while expanding one module to illustrate representative patch-, module-, and parameter-level information. Repeated elements and lower-level fields are elided for readability. The complete version of this snapshot is available in the companion repository [29].

```
{
  "type": "patch_snapshot",
  "schemaVersion": 1,
  "revision": 1,
  "layout": {
    "coordinateSpace": "vcv_rack_pixels",
    "origin": "top_left"
  },
  "patchBounds": {
    "pos": {"x": 30000.0, "y": 38000.0},
    "size": {"x": 915.0, "y": 760.0}
  },
  "modules": [
    {...},
    {
      "moduleInstanceId": 6,
      "pluginSlug": "Fundamental",
      "modelSlug": "ADSR",
      "modelName": "ADSR EG",
      "moduleBox": {
        "pos": {"x": 30645.0, "y": 38380.0},
        "size": {"x": 135.0, "y": 380.0}
      },
      "params": [
        {
          "paramId": 0,
          "label": "Attack",
          "role": "slider",
          "behavior": "continuous",
          "paramBox": {"pos": {...}, "size": {...}},
          "value": 0.388,
          {...}
        },
        {...},
        {
          "paramId": 4,
          "label": "Attack CV",
          "role": "knob",
          "behavior": "continuous",
          {...}
        },
        {...},
        {
          "paramId": 8,
          "label": "Push",
          "role": "button",
          "behavior": "momentary",
          {...}
        },
        {...}
      ]
    },
    {...}
  ]
}
```

Fig. 4. Selected structural excerpt from a VeryVCV Rack patch snapshot.

C. Communication and Networking Layer

The *Connection Layer* defined by the model is implemented in VeryVCV Rack through two complementary channels over the same local Wi-Fi network. As shown in Figure 5, the *Description* flow transfers JSON patch snapshots over HTTP, with snapshot availability signaled through a lightweight OSC notification, while the bidirectional *Updates* flow uses OSC over UDP for runtime control and state messages.

The snapshot channel is event-driven. When the user presses the dedicated button on the Bridge module, the Bridge scans the current VCV Rack patch, generates a new patch snapshot, serializes it as JSON, and stores it in a local HTTP cache. Instead of pushing the full JSON description through OSC, the Bridge sends a lightweight snapshot-available notification containing the current patch-session identifier and an incremental revision number. The MR client compares this revision with that of its current snapshot and retrieves the updated JSON through a read-only HTTP request only when a newer description is available.

At runtime, parameter values are checked continuously during the Rack UI update loop. When the Bridge detects a parameter change in the live VCV Rack patch, it sends a `/veryvcv/param/update` message containing the patch-session identifier, revision number, target module and parameter identifiers, and value. Conversely, manipulation of a spatial control generates a `/veryvcv/param/set` message identifying the active patch session, target parameter, and value to apply.

Revision numbers associate runtime updates with the corresponding structural description, while VCV Rack retains authority over parameter state. When a `/veryvcv/param/set` message is received, the Bridge verifies the active patch session, resolves the target parameter, applies the requested value, and reads back the value accepted by VCV Rack. This readback is cached as the last propagated state, preventing the subsequent update cycle from echoing the same change back to the MR client.

interactive MR scene in which VCV Rack modules are reconstructed as spatial panels populated with hand-interactable controls.

When a new snapshot is retrieved through the HTTP channel, the client parses the JSON into a runtime model reflecting the functional structure described in Section IV-B. The MR scene is then generated automatically by combining this model with a library of reusable MR components indexed by interface role. For each VCV Rack module, the client creates a spatial panel whose dimensions and position are derived from the exported Rack layout. Interface elements are instantiated with position and scale derived from the snapshot. Interactive controls are additionally assigned the module and parameter identifiers required for synchronization and initialized from the corresponding normalized parameter value. This role-based mapping implements the *Spatial Mapping* stage of the proposed model and allows heterogeneous VCV Rack modules to be reconstructed from a common description schema without dedicated MR mappings for individual modules.

Hand tracking is used as the primary interaction modality, relying on Meta XR Interaction SDK components for hand sensing and object manipulation. Buttons use direct poke interaction, while sliders are manipulated by grasping their handle and moving it along a constrained linear axis. Knobs use a hybrid manipulation strategy combining rotation of the hand around the control axis with vertical and horizontal displacement relative to the knob, providing translational input in addition to wrist rotation. Manipulation of these controls generates runtime parameter messages to the Bridge, while parameter updates received from VCV Rack are applied to the corresponding MR components. To avoid interaction conflicts, incoming updates are temporarily prevented from modifying a component while it is being manipulated locally. A demonstration video showing the three implemented interaction techniques is available in the companion repository [29].

V. PRELIMINARY SYSTEM CHARACTERIZATION

To characterize the behavior and current constraints of the VeryVCV Rack prototype, we examined three implementation-level aspects: description scalability, observable send-to-visible-response offset, and interface-description coverage across heterogeneous VCV Rack modules. The measurements characterize the reference implementation under the hardware and network conditions described below and are not intended as hardware-independent benchmarks of the proposed integration model. This system-level characterization does not assess usability, effectiveness, or musical suitability. The audio path is also outside the scope of the present measurements and the synchronization measure reported below does not include action-to-sound latency.

All tests were conducted with VCV Rack running on a MacBook Pro with an M4 Pro processor and 24 GB of memory and the MR client running on a Meta Quest 3. Both devices were connected to the same Wi-Fi hotspot at an approximate distance of 1 m. The connection used 802.11n in the 2.4-GHz

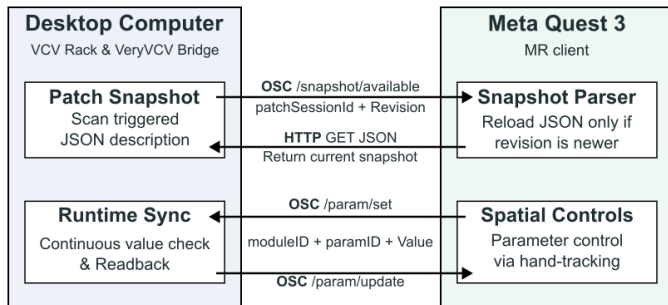


Fig. 5. Communication layer of VeryVCV Rack, separating patch-description transfer from bidirectional runtime parameter updates.

D. Quest MR Client

The Quest-side client is implemented as a standalone Unity application for the Quest 3. It retrieves the structured patch snapshot exposed by the Bridge and translates it into an

band, on channel 8 with a 20-MHz channel width. Measurements were conducted in a non-isolated wireless environment with multiple surrounding Wi-Fi networks.

A. Description Scalability

Description scalability was evaluated by measuring how snapshot generation and reconstruction behave as patch size increases. Five patches were constructed with a single VeryVCV Bridge and progressively larger subtractive-synthesis configurations based on the same small set of module types, including VCO, VCF, VCA, and Audio Out modules. The resulting patches ranged from 5 to 80 modules.

For each configuration, 30 scan-and-reconstruction cycles were performed. To avoid clock synchronization between the desktop computer and headset, timings were measured independently on each device: the Bridge logged the VCV Rack-side stages, while the Quest client measured its processing after receiving the snapshot-availability message. Quest timings were subsequently returned to the Bridge through OSC and stored in the same CSV log. Individual stage timings were retained for each run, while Table I reports aggregate Bridge- and Quest-side measures for compactness. Complete per-run logs, including the disaggregated timings, are available in the companion repository [29].

TABLE I
SNAPSHOT SCALABILITY ACROSS INCREASINGLY LARGE PATCHES
CONSTRUCTED FROM THE SAME SMALL SET OF MODULE TYPES. TIMING
VALUES ARE REPORTED AS MEAN $\pm$ STANDARD DEVIATION OVER 30
RUNS.

Patch	Modules	Elements	JSON (KB)	Bridge (ms)	Quest (ms)
P1	5	41	19.46	1.22 $\pm$ 0.18	62.44 $\pm$ 16.59
P2	10	102	48.54	1.77 $\pm$ 0.31	120.09 $\pm$ 17.47
P3	20	206	97.61	3.19 $\pm$ 0.53	353.37 $\pm$ 77.03
P4	40	417	196.98	3.90 $\pm$ 0.57	767.96 $\pm$ 103.12
P5	80	837	394.81	5.33 $\pm$ 0.53	2255.06 $\pm$ 225.79

In Table I, *Elements* is the sum of module panels, controls, and jacks. *Bridge* covers patch scanning, JSON serialization, HTTP-cache update, and snapshot notification, whereas *Quest* aggregates HTTP retrieval, JSON parsing, and client-side instantiation.

Bridge-side processing increased from 1.22 to 5.33 ms across the tested configurations, whereas aggregate Quest-side time increased from 62.44 to 2255.06 ms. Quest-side loading and reconstruction grew more than proportionally with the number of exported elements over the tested range: from P2 to P5, the element count increased by approximately eight times, while aggregate Quest time increased by almost nineteen times. The disaggregated logs indicate that this growth is driven primarily by client-side instantiation, which increased from 63.88 ms in P2 to 1721.21 ms in P5 and accounted for approximately 76% of the Quest-side total in the largest configuration. Over the same interval, HTTP retrieval increased from 54.50 to 525.64 ms, while JSON parsing remained comparatively small, reaching 7.22 ms in P5. The comparatively small parsing time and increasing contribution

of the instantiation stage suggest that the observed Quest-side growth is associated primarily with construction of the reconstructed MR scene rather than description parsing or network transfer.

The tested patches were deliberately constructed from the same small set of module types, thereby isolating increasing element count under a relatively homogeneous workload rather than interface heterogeneity. During manual inspection, the largest patch remained operable, although hand-tracking interaction appeared less smooth. Since sustained frame rate and hand-tracking stability were not instrumented, this observation remains qualitative and does not establish runtime performance at this scale. Further characterization should therefore include heterogeneous large patches and sustained runtime-performance measurements.

B. Observable Send-to-Visible-Response Offset

To observe runtime synchronization behavior, we used a video-based method inspired by camera-based end-to-end measurement approaches for virtual environments and interactive systems [31], [32]. A 60 fps recording was captured from the Quest viewpoint, framing both an MR button and the corresponding module on the VCV Rack screen. The tested button was instrumented so that its visual state turned green in the same client update that sent the corresponding OSC parameter-control message. In VCV Rack, the controlled parameter was connected to a visible scope, allowing the resulting trigger to be observed on the computer screen through the Quest passthrough view. This provided a common video reference for comparing the client-side send marker with the visible desktop response without requiring clock synchronization between the headset and desktop computer.

The recording was inspected frame by frame over 100 button presses. For each trial, we identified the first video frame in which the MR button appeared green and the first frame in which the VCV Rack scope showed the corresponding trigger signal. The observed offset was defined as the difference between these frame indices. In 8 trials, both events were visible in the same frame; in 74 trials, the trigger appeared one frame after the send marker; and in 18 trials, it appeared two frames later. At 60 fps, each frame corresponds to approximately 16.67 ms. The largest observed offset was therefore two frames, corresponding to approximately 33.3 ms. The mean observed frame offset was 1.10 frames, corresponding to approximately 18.3 ms when expressed at the 60-fps frame interval.

These values should not be interpreted as precise latency measurements. The method cannot resolve the relative timing of events occurring within the same video frame, and the desktop response is observed through the Quest passthrough and capture pipeline whereas the MR send marker is rendered locally, so the two visual events do not follow identical acquisition paths. The test was also limited to a button because it provides a discrete and visually unambiguous event marker, unlike continuous knob or slider manipulation. The measurement therefore characterizes only the observable frame

offset between a client-side parameter-message send event and the corresponding visible response in the desktop VCV Rack interface under the tested configuration.

C. Interface-Description Coverage

To complement the scalability measurements, we inspected the structured descriptions produced by the Bridge for 100 VCV Rack modules from 20 different brands or developers. The set was purposively selected to include modules familiar to the authors from regular VCV Rack use while covering heterogeneous interface designs. For each module, the exported JSON snapshot and resulting MR scene were compared with the source interface. Table II summarizes these observations; individual modules may contribute to more than one issue category.

Across the inspected set, 52 modules were reconstructed without any identified module-specific mismatch. For the remaining modules, issues ranged from label differences and unsupported visualization elements to mismatches affecting control behavior or state-dependent reconstruction. Standard VCV Rack interface components, including classes derived from them through C++ inheritance, were generally well reconstructed from the exported roles, values, positions, and interaction modes. Fully custom elements were less predictable: 12 modules contained at least one unresolved role or behavior, corresponding to 58 of the 1852 exported parameters, and would require explicit annotations or module-specific adapters for more reliable reconstruction.

Several recurring limitations were also observed. State-dependent visibility is not currently represented in the description: 12 modules contained overlapping parameter boxes, 3 of which also contained unresolved roles or behaviors. Manual inspection confirmed state-dependent control mismatches in 8 modules, where alternative controls occupying the same panel position could be instantiated simultaneously in MR. Control orientation is likewise absent from the current description. Seven modules contained non-vertical sliders, accounting for 21 of the 101 sliders in the inspected set, whose orientation cannot currently be distinguished by the MR client as it is not currently represented in the description.

Interaction behavior and labeling produced additional mismatches. Eleven modules contained at least one button or switch whose reconstructed interaction did not correspond to the source behavior. Control labels are derived from developer-provided component or parameter names and therefore do not necessarily reproduce the visible panel text; label mismatches were observed in 28 modules. Dense layouts can additionally cause otherwise valid labels to overlap in the reconstructed interface.

Visual identity is also only partially preserved. The current description does not transfer source panel graphics or color schemes, so reconstructed modules use a generic visual style while retaining their exported dimensions, layout, and control organization. Displays and other visualization elements likewise fall outside the current parameter-oriented schema and were present in 18 of the inspected modules. Broader

TABLE II
INTERFACE-DESCRIPTION COVERAGE ACROSS 100 VCV RACK MODULES
FROM 20 BRANDS.

Measure	Count
<i>Sample and description-level indicators</i>	
Number of modules	100
Exported parameters	1852
Modules with ≥ 1 unresolved role/behavior	12
Parameters with unresolved role/behavior	58/1852
Modules with overlapping parameter boxes	12
Modules with unresolved role/behavior and overlap	3
Modules containing non-vertical sliders	7
Non-vertical sliders	21/101 sliders
<i>Observed reconstruction</i>	
Modules reconstructed without identified mismatch	52
Modules with label mismatch	28
Modules with button/switch interaction mismatch	11
Modules with state-dependent control mismatch	8
Modules containing unsupported displays or visualizations	18

coverage across the VCV Rack ecosystem would therefore require additional description mechanisms for custom controls, visibility, orientation, interaction modes, visual feedback, and source-side graphical information.

VI. DISCUSSION

The preliminary characterization supports the technical feasibility of VeryVCV Rack as an instantiation of the proposed integration model while clarifying its current constraints. The scalability measurements showed that Bridge-side snapshot generation remained comparatively small as patch size increased, whereas Quest-side scene instantiation became the dominant contribution. This suggests that the main scalability constraint of the current implementation lies in client-side scene materialization rather than in generating or parsing the structured description. Support for larger patches should therefore focus primarily on optimizing MR scene instantiation.

The interface-description inspection further qualifies this result. Structured patch descriptions provided a workable basis for XR reconstruction when the source environment exposed sufficient information about interface elements and their interaction properties. Standard VCV Rack components could generally be reconstructed from the available data, whereas limitations emerged when relevant properties were absent or could not be inferred reliably. Extending this approach across different systems therefore requires descriptions that communicate not only parameter values, but also what interface elements represent, how they behave, how they relate to one another, and how they are spatially organized. A sufficiently consistent and interpretable description vocabulary would allow this information to function as shared interaction knowledge between source environments and XR clients.

A. Applicability Beyond VCV Rack

The VeryVCV Rack implementation benefits from VCV Rack being a favorable source environment. Its plugin architecture exposes modules, parameters, graphical components, and layout through Rack-native abstractions, making it possible to inspect the live patch and export a structured snapshot.

Other music-software environments may expose automation parameters without layout information, graphical components without control roles, or plug-in interfaces without exposed visibility and state information. Description-driven translation should therefore not be understood as an automatic conversion mechanism for arbitrary music software.

The integration model is not conceptually tied to VCV Rack, but applying it to other environments would require an interface-description contract between the source application and the XR client. The underlying principle of such structured interface contracts is well established in web and accessibility architectures, where mechanisms such as the DOM and WAI-ARIA expose interface structure and semantics independently of their visual rendering [14], [27]. In the present context, such a contract would specify the information required to interpret and spatialize interface elements, including parameter associations, roles, labels, layout relationships, interaction modes, and visibility states. Depending on the source environment, this information could be provided through native APIs, plug-in extensions, adapter layers, or dedicated metadata formats.

B. Toward XR-Aware Music Software

The current prototype adopts a conservative form of translation: standard VCV Rack controls are reconstructed as spatial MR components that remain recognizable with respect to the original two-dimensional interface. This supports continuity between desktop and headset interaction, but does not exploit the full range of spatial interaction possibilities offered by XR. Future implementations could expose richer descriptions that allow alternative spatial representations while retaining a conventional desktop interface.

For example, three related parameters represented as separate knobs on the desktop could be exposed by the source environment not only as individual controls, but also as members of an explicit semantic group. An XR-aware description could then specify that this group admits an alternative spatial representation, such as a three-dimensional parameter space in which the coordinates of a movable point control the three corresponding values. The individual desktop controls would remain available and synchronized, preserving their identities and parameter associations, while the XR representation would provide an additional way of manipulating the same underlying structure. In this sense, the spatial representation would not result from an arbitrary reinterpretation by the XR client, but from richer interaction knowledge explicitly exposed by the source environment. Correspondence with the source interface would therefore be preserved at the level of parameter identity, grouping, and behavior, even when the spatial affordance does not reproduce the original panel one-to-one.

XR-aware representations would therefore not replace the description-driven model, but extend the information exposed by the source environment from individual control roles toward higher-level semantic groupings and alternative spatial mappings. Description-driven translation can consequently be viewed as a continuum, from reconstruction of existing controls to richer representations based on explicitly described

relationships among parameters. This direction requires additional developer support while preserving the central architectural property of the model: the source environment retains responsibility for computation and parameter state, and XR provides an alternative interaction surface derived from an explicit interface description.

C. Implications for the Musical Metaverse

For Musical Metaverse research, the proposed approach represents an integration model rather than a migration model. Integration concerns both where musical computation remains and what information is shared across environments. Rather than transferring the complete musical system into XR, VeryVCV Rack preserves the desktop environment as the computational and workflow core while exposing the interaction knowledge required by an XR client to construct an alternative synchronized interface. In this sense, the approach extends interoperability as message exchange toward interoperability as shared interaction knowledge: source environments and XR clients exchange not only runtime control values, but also structured information about interface elements, roles, states, relationships, and interaction behavior.

This architectural continuity may be relevant to workflows that already depend on established software, plug-ins, multi-channel audio interfaces, MIDI controllers, or other hardware connected to the desktop environment, but it also introduces trade-offs. The XR layer does not replace the host computer and therefore requires both the existing music software setup and an XR headset. Moreover, the current prototype does not provide a shared multi-user environment, networked audio transport, or distributed musical state across performers. In a broader Musical Metaverse scenario, preserving audio computation in the source environment would require separate mechanisms for transmitting or sharing audio when remote participants need access to the same sonic output. The present contribution should therefore be understood as addressing software-to-XR interface integration rather than as a complete Musical Metaverse infrastructure.

Within this scope, XR is treated as a situated interaction layer rather than a replacement for the desktop interface. Editing, patch construction, configuration, and routing can remain available through the original application, while spatial organization, embodied manipulation, and coexistence with physical instruments provide possible contexts for XR interaction. The present study does not establish whether these affordances provide an advantage over desktop, touch-based, or hardware interfaces. VeryVCV Rack provides an initial case study of this integration strategy in modular synthesis and a basis for investigating how established music-software environments can participate in hybrid MR/XR workflows without migrating their complete functionality into the immersive client.

VII. CONCLUSION AND FUTURE WORK

This paper presented VeryVCV Rack as a prototype instantiation of a description-driven XR integration model for extending VCV Rack interfaces into mixed reality. The approach

preserves the desktop environment as the computational and workflow core while exposing structured interaction information that an XR client can use to reconstruct and synchronize a spatial interface. The preliminary system characterization supports the technical feasibility of this approach under the tested conditions and highlights the importance of sufficiently rich source-side interface descriptions for reliable reconstruction.

Future work will further characterize runtime behavior, including action-to-sound latency, sustained frame rate, hand-tracking stability, and performance with larger and more heterogeneous patches. The prototype will also be extended toward richer interface descriptions, including support for source panel graphics and color schemes, state-dependent control visibility, control orientation, dynamic visual feedback, and patch-cable creation and editing. At the design level, XR-aware modules could expose higher-level semantic relationships and alternative spatial mappings while retaining their conventional VCV Rack representations. Finally, user studies with musicians and live-electronics performers will be required to assess usability, musical suitability, workflow implications, and the contexts in which XR interaction provides meaningful benefits. Together, these directions can further clarify the role of description-driven integration in hybrid Musical Metaverse workflows that retain established music-software ecosystems.

ACKNOWLEDGMENT

Giuseppe Bergamino's PhD scholarship is funded by the Marche Region within the framework of the "Innovative PhD Programmes", under the Regional Programme of the European Social Fund Plus (PR FSE+) 2021–2027.

REFERENCES

- [1] L. Turchet, R. Hamilton, and A. Çamci, "Music in extended realities," *IEEE Access*, vol. 9, pp. 15 810–15 832, 2021.
- [2] L. Turchet, "Musical Metaverse: Vision, opportunities, and challenges," *Personal and Ubiquitous Computing*, vol. 27, no. 5, pp. 1811–1827, 2023.
- [3] A. Boem, M. Tomasetti, and L. Turchet, "Issues and challenges of audio technologies for the musical metaverse," *Journal of the Audio Engineering Society*, vol. 73, no. 3, pp. 94–114, 2025.
- [4] M. Leman, *Embodied Music Cognition and Mediation Technology*. MIT Press, 2007.
- [5] T. Magnusson, "Of epistemic tools: Musical instruments as cognitive extensions," *Organised Sound*, vol. 14, no. 2, pp. 168–176, 2009.
- [6] A. Kitzmann and C. Thorén, "The modular journey: Uncovering analogue aesthetics in digital landscapes," *Organised Sound*, vol. 27, no. 1, pp. 44–54, 2022.
- [7] G. Wakefield, M. Palumbo, and A. Zonta, "Affordances and constraints of modular synthesis in virtual reality," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2020, pp. 547–550.
- [8] L. Zeller and H. Barfuss, "OpenSoundLab: A virtual sound laboratory for the arts," in *Proceedings of the 17th International Audio Mostly Conference*, ser. AM '22. Association for Computing Machinery, 2022, pp. 159–162.
- [9] D. Johnson, D. Damian, and G. Tzanetakis, "OSC-XR: A toolkit for extended reality immersive music interfaces," in *Proceedings of Sound and Music Computing Conference*, 2019, pp. 202–209.
- [10] G. A. Giudici, D. Stefani, A. Boem, and L. Turchet, "Connecting MIDI interfaces to the musical metaverse," in *Proceedings of the 6th IEEE International Symposium on the Internet of Sounds*, 2025, pp. 1–9.
- [11] M. Buffa, D. Girard, and A. Hofr, "Using web audio modules for immersive audio collaboration in the musical metaverse," in *Proceedings of the 2024 IEEE 5th International Symposium on the Internet of Sounds*, 2024, pp. 1–10.
- [12] M. Buffa, M. Winckler, Q. Escobar, S. Demont, A. Hofr, and A. Mir-Sadjadi, "Interactive audio sculpting: Plugin customization and UI affordances in immersive environments," in *Proceedings of the 6th IEEE International Symposium on the Internet of Sounds*, 2025, pp. 1–10.
- [13] D. Dziwis, "PdXR - the evolution of pure data into the metaverse," in *Proceedings of the International Computer Music Conference (ICMC 2023)*, Shenzhen, China, 2023, pp. 52–58.
- [14] W3C, "Accessible rich internet applications (WAI-ARIA) 1.2," W3C Recommendation, 2023. [Online]. Available: <https://www.w3.org/TR/wai-aria-1.2/>
- [15] L. Gabrielli, *Developing Virtual Synthesizers with VCV Rack*. Focal Press, 2020.
- [16] L. Zeller and C. E. Leisi, "Modular observers: OpenSoundLab and PatchWorld as case studies for emerging practices of modular synthesis in extended realities," *Organised Sound*, 2026, first View.
- [17] VCV Rack, "VCV Rack 2: Virtual Eurorack Studio," <https://vcvrack.com/Rack>, 2026, accessed: 25 Jun 2026.
- [18] —, "VCV Library," <https://library.vcvrack.com/>, 2026, accessed: 25 Jun 2026.
- [19] —, "Plugin API Guide," <https://vcvrack.com/manual/PluginGuide>, 2026, accessed: 25 Jun 2026.
- [20] C. Betancur Gutiérrez, "Umbilical: A hybrid system for creating custom modular midi controllers – a eurorack/vcv rack case study," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, London, United Kingdom, 2026, pp. 979–982.
- [21] M. Palumbo, A. Zonta, and G. Wakefield, "Modular reality: Analogues of patching in immersive space," *Journal of New Music Research*, vol. 49, no. 1, pp. 8–23, 2020.
- [22] L. Zeller and A. Bauer, "The walkable instrument: Modular patches as entangled environments in OpenSoundLab," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2025.
- [23] L. Turchet, P. Bouquet, A. Molinari, and G. Fazekas, "The smart musical instruments ontology," *Journal of Web Semantics*, vol. 72, p. 100687, 2021.
- [24] M. Hughes and A. Johnston, "URack: Audio-visual composition and performance using Unity and VCV Rack," in *Proceedings of the International Conference on New Interfaces for Musical Expression*, 2020, pp. 321–322.
- [25] MDN Web Docs, "WAI-ARIA Roles," 2025, accessed: 25 Jun. 2026. [Online]. Available: <https://developer.mozilla.org/en-US/docs/Web/Accessibility/ARIA/Reference/Roles>
- [26] JUCE, "AccessibilityHandler Class Reference," https://docs.juce.com/master/classjuce_1_1AccessibilityHandler.html, 2025, accessed: 25 Jun. 2026.
- [27] WHATWG, "DOM Standard," <https://dom.spec.whatwg.org/>, 2026, accessed: 25 Jun 2026.
- [28] VCV Rack, "VCV API," Developer documentation, 2026, accessed: 25 Jun 2026. [Online]. Available: <https://vcvrack.com/docs-v2/>
- [29] G. Bergamino, "Companion GitHub repository," <https://github.com/GiuseppeBergamino/A-Description-Driven-XR-Translation-of-VCV-Rack>, 2026.
- [30] Meta, "Develop Unity Apps for Meta Quest VR Headsets," <https://developers.meta.com/horizon/documentation/unity/unity-development-overview/>, 2025, accessed: 25 Jun 2026.
- [31] D. He, F. Liu, D. Pape, G. Dawe, and D. Sandin, "Video-based measurement of system latency," in *Proceedings of the International Immersive Projection Technology Workshop*, Ames, IA, USA, 2000.
- [32] S. Friston and A. Steed, "Measuring latency in virtual environments," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, no. 4, pp. 616–625, 2014.