```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ieee1599 SYSTEM "http://www.lim.di.unimi.it/IEEE/ieee1599.dtd">
<ieee1599 version="1.0" creator="Luca A. Ludovico">
    <general>
        <description>
            <main_title>Gottes Macht und Vorsehung</main_title>
            <author type="composer">Beethoven, Ludwig van</author>
            <author type="poet">Christian Fürchtegott Gellert</author>
            <number>5</number>
            <work_title>6 Geistliche Lieder Von Gellert</work_title>
            <work_number>op. 48</work_number>
        </description>
    </general>
    <structural>...</structural>
    <logic>
        <spine>
            <event id="clef_1" timing="0" hpos="1"/>
            <event id="timesig_1" timing="0" hpos="5"/>
            <event id="keysig_1" timing="0" hpos="NULL"/>
            <event id="v1_e0" timing="0" hpos="6"/>
            <event id="v1_e1" timing="4" hpos="6"/>
            ...
        </spine>
        <los>
            <agogics event_ref="timesig_1">Mit Kraft und Feuer.</agogics>
            <staff_list>
                <staff id="staff_1">
                    <clef shape="G" event_ref="clef_1" staff_step="2"/>
                    <time_signature event_ref="timesig_1">
                        <time_indication num="2" den="2" vtu_amount="8"/>
                    </time_signature>
                    <key_signature event_ref="keysig_1">
                        <flat_num number="0"/>
                    </key_signature>
                </staff>
                ...
            </staff_list>
            <part id="soprano">
                <voice_list>
                    <voice_item id="soprano_voice1" staff_ref="staff_1"/>
                </voice_list>
                <measure number="1">
                    <voice voice_item_ref="soprano_voice1">
                        <rest event_ref="v1_e0" staff_ref="staff_1">
                            <duration num="1" den="2"/>
                        </rest>
                        <chord event_ref="v1_e1">
                            <duration num="1" den="2"/>
                            <notehead staff_ref="staff_1">
                                <pitch step="C" octave="6"/>
                            </notehead>
                        </chord>
                    </voice>
                </measure>
                ...
            </part>
            <part id="piano">...</part>
            <lyrics part_ref="soprano" voice_ref="soprano_voice1">
                <syllable start_event_ref="v1_e1" end_event_ref="v1_e1">Gott</syllable>
                <syllable start_event_ref="v1_e2" end_event_ref="v1_e2">ist</syllable>
                ...
            </lyrics>
        </los>
    </logic>
    <notational>
        <graphic_instance_group description="Finale">
            <graphic_instance file_name="gottes/gottes_finale.tif"
                file_format="image_tiff" encoding_format="image_tiff" position_in_group="1" measurement_unit="pixels">
                <graphic_event event_ref="v1_e0" upper_left_x="646" upper_left_y="616" lower_right_x="702" lower_right_y="648"/>
                <graphic_event event_ref="v1_e1" upper_left_x="764" upper_left_y="608" lower_right_x="818" lower_right_y="712"/>
                ...
            </graphic_instance>
        </graphic_instance_group>
        <graphic_instance_group description="Dover">...</graphic_instance_group>
    </notational>
    <performance>
        <midi_instance file_name="gottes/gottes.mid" format="1">
            <midi_mapping part_ref="soprano" track="1" channel="1">
                <midi_event_sequence division_type="timecode" division_value="1024" measurement_unit="ticks">
                    <midi_event event_ref="v1_e0" timing="0"/>
                    ...
                </midi_event_sequence>
            </midi_mapping>
        </midi_instance>
    </performance>
    <audio>
        <track file_name="gottes/gottes_midi.mp3" file_format="audio_mp3" encoding_format="audio_mpeg">
            <track_indexing timing_type="seconds">
                <track_event start_time="0.00" event_ref="v1_e0"/>
                <track_event start_time="1.00" event_ref="v1_e1"/>
                ...
            </track_indexing>
        </track>
    </audio>
</ieee1599>
```

Figure 3: An XML skeleton illustrating a typical IEEE 1599 document's overall structure.

for improving accessibility in music interaction, particularly for individuals with physical and cognitive impairments. Thanks to its multilayer structure and support for synchronized, multimodal content, IEEE 1599 enables tailored user experiences that can accommodate diverse abilities and needs. By decoupling the various dimensions of musical information, it becomes possible to provide alternative or complementary representations of the same musical content. This flexibility opens new possibilities for inclusive archival access, making music more reachable to those who face sensory, motor, or cognitive barriers. IEEE 1599's capacity to host multiple instances within each layer supports personalization and adaptability, essential principles in inclusive design. For individuals with visual impairments, traditional sheet music, typically rendered as printed notation, presents an obvious barrier.

IEEE 1599 addresses this by allowing the symbolic layer to be interpreted and rendered in alternative modalities. For example, the symbolic content can be sonified through screen reader compatible software or converted into Braille music notation via specialized transcription tools. Moreover, since the format supports synchronized playback of audio and symbolic data, it becomes possible to navigate a score using auditory cues alone, with the system announcing measure numbers, note names, dynamics, or other relevant annotations as the music progresses. This facilitates both passive listening and active learning, and it enables users with visual impairments to engage with music independently and interactively. In addition to supporting users with complete vision loss, IEEE 1599 can accommodate the needs of individuals with low vision or color vision deficiencies. As an example, it is possible to dynamically adjust the visual representation of the score without compromising the integrity of the underlying musical data. Users with low vision, for instance, may benefit from high-contrast rendering, enlarged notation, or simplified layouts tailored to their specific visual capabilities. Similarly, for colorblind users, color schemes used in annotated scores or educational overlays, such as pitch-color mappings, can be customized to ensure perceptual accessibility, avoiding problematic color pairs (e.g., red green) and replacing them with alternatives or patterns. For individuals with aural impairments, especially those who are deaf or hard of hearing, traditional music experiences centered on listening are often inaccessible or limited. However, IEEE 1599 offers tools to reframe musical content in ways that are not exclusively reliant on auditory perception. By leveraging the symbolic and graphical layers, users can engage with music visually through animated scores, gesture-based representations, or real-time visualizations of rhythm, pitch, and dynamics. A deaf user might follow a synchronized graphic score while receiving haptic feedback for rhythmic elements, or watch an animated cursor highlighting the current note in the symbolic layer as it progresses in time, or enjoy an alternative graphical representation based on the logic layer (see Figure 4). Furthermore, the lyrics sub-layer can be rethought to enrich the musical experience with textual annotations, descriptions of expressive intent, or

contextual information about the composition.

In this way, IEEE 1599 promotes a multimodal approach to music that expands participation beyond hearing-based modalities, supporting inclusive music appreciation and education for the aurally impaired. Motor impairments can significantly limit a person's ability to interact with traditional musical instruments, printed scores, or even standard user interfaces for digital music systems. IEEE 1599 helps address these barriers by enabling alternative modes of interaction with music that do not rely on fine motor skills. For instance, its multilayer structure allows for simplified or adaptive control interfaces (e.g., switches, eye-tracking devices, and single-button systems) that trigger synchronized audio, symbolic, and visual outputs. A user may, for example, navigate through a piece of music using a single switch input, receiving auditory feedback from the audio layer while simultaneously viewing note information or performance cues from the graphical or symbolic layers [5]. Moreover, the integration of logic and structure layers can enable the creation of simplified versions of a composition suited to the user's physical capabilities while preserving musical coherence. In educational and rehabilitative contexts, this can empower users with motor impairments to explore, understand, and even perform music in meaningful and personalized ways. For individuals with cognitive impairments, such as difficulties with memory, attention, language processing, or executive functioning, engaging with music can present unique challenges. Traditional scores may be too dense or complex, and standard presentation methods may not align with their expectations. IEEE 1599 provides a flexible framework to mediate this interaction by allowing the creation of simplified, multimodal, and customizable representations of musical content. For example, the symbolic layer can be used to extract and highlight structural elements, like repeating sections, thematic material, or simplified melodies. Furthermore, an ad-hoc encoding can support textual cues, icon-based annotations, or step-by-step instructions adapted to the user's cognitive profile. Finally, the temporal synchronization across layers makes it possible to create guided listening experiences, where visual cues, audio playback, and interactive elements are tightly coordinated. These features reduce cognitive overload and can help maintain focus and motivation. As such, IEEE 1599 facilitates the design of music-based applications that are not only more accessible but also more effective for users with diverse cognitive needs. Inclusivity is also promoted through the format's neutrality with respect to representation hierarchies: no single layer is privileged over others. This allows for the coexistence of Western classical notation with alternative notational systems, oral traditions, or contemporary multimedia compositions. Moreover, the ability to integrate culturally specific metadata and diverse performance practices aligns with the goals of inclusive heritage preservation.

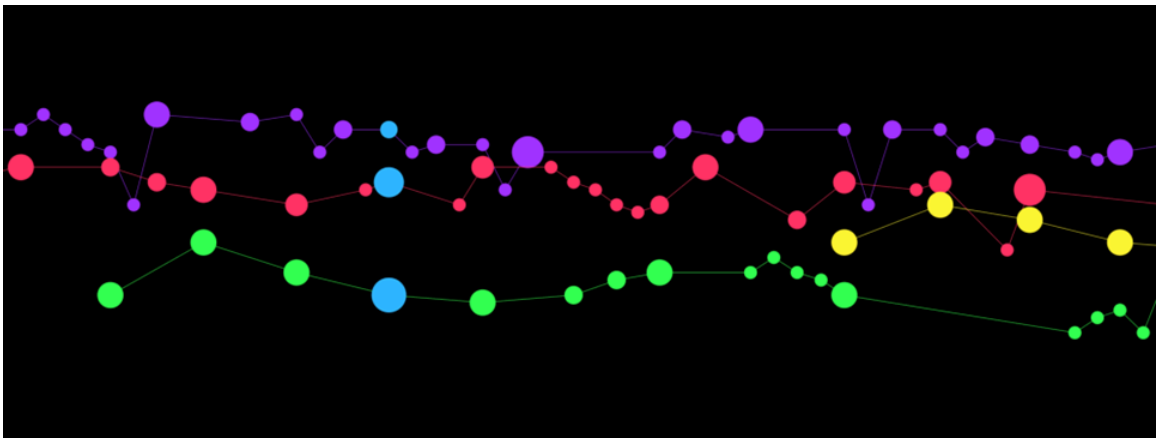

Figure 4: An example of an alternative graphical representation for logic-layer content.

Chapter 3

Accessible Digital Music Instruments

3.1 Definitions

3.1.1 Digital Music Instruments

A fundamental distinction in the organology of electronic music is the structural decoupling of the control interface from the sound generation unit. Unlike traditional acoustic instruments, where the excitation mechanism (e.g. the plucking of a string or the vibration of a reed) is physically and acoustically coupled to the resonating body, Digital Musical Instruments (DMIs) consist of two independent components: a gestural controller (input) and a sound engine (output). These two distinct subsystems are connected via a mapping layer, which translates physical control signals into musical parameters [48].

In the context of HCI, a DMI can be defined as a system where the user interface is separate from the application logic. The gestural controller captures human physical activity—such as pressure, velocity, position, or motion—using various sensor technologies. This data is then digitized and transmitted to the sound engine, which may differ significantly in nature, ranging from sample-based synthesis to physical modeling or granular synthesis [16]. The relationship between the input gesture and the auditory output is not fixed by the laws of physics but is instead defined by the designer through algorithmic mapping strategies.

This modular architecture, often referred to as "Digital Lutherie" [49], allows for an infinite variety of relationships between gesture and sound. Although in the past it was common to attribute a "one-to-one" mapping for the mimic of acoustic instruments (e.g., one key press triggers one specific pitch), studies have shown[50]

that even in those situations, complex mapping strategies do perform better than one-to-one relationships[51]. In fact, "one-to-many" or "many-to-one" mappings allow for complex, multilayered interactions where a single gesture might control multiple synthesis parameters simultaneously, or conversely, where multiple sensor inputs are required to produce a single sonic event [52].

The communication between the controller and the sound engine is typically mediated by standard protocols, most notably MIDI (Musical Instrument Digital Interface) and OSC (Open Sound Control). While the controller is responsible for the haptic and ergonomic aspects of the performance—determining how the musician physically engages with the instrument—the protocol ensures that these physical interactions are quantified and transmitted as data. This separation is the technological prerequisite that enables the existence of Accessible Digital Musical Instruments (ADMIs): because the control interface is not permanently bound to the sound source, it can be redesigned, adapted, or entirely replaced to accommodate non-normative bodies without compromising the quality or complexity of the sound produced.

3.1.2 Accessible Digital Music Instruments

Inclusive music technologies aim to enable individuals of all abilities to participate in music-making by overcoming physical, cognitive, or sensory barriers. Traditional instruments often assume normative motor and sensory functions, making them inaccessible for many. This challenge has led to the emergence of ADMIs, custom-designed systems that leverage adaptive hardware and multimodal interfaces to broaden participation in musical expression [53, 54].

Research in ADMIs includes a diverse range of interaction paradigms: gaze-based instruments such as EyeHarp [55], gesture-based systems using Leap Motion or inertial sensors [56, 57, 58], and touchless controllers that utilize proximity sensing or computer vision [59]. Gaze-controlled systems have shown particular promise for users with limited motor ability, enabling control over pitch, timing, and dynamics through eye movements [60, 61, 62]. Gesture-based systems similarly offer expressive input while removing the need for precise contact or fine motor control.

Beyond input modalities, the field has embraced user-centered and participatory design approaches, recognizing that accessibility must be tailored and co-developed with end users [63, 64]. These approaches ensure that musical tools are not only technically accessible but also creatively empowering and socially meaningful.

MIDI (Musical Instrument Digital Interface) has historically played a critical role in inclusive music-making. Its original specification, MIDI 1.0, established in 1983, provided a standardized protocol for musical communication across devices [65, 66]. This allowed alternative controllers, such as sip-and-puff devices, custom button grids, or eye trackers, to interface with commercial DAWs and sound modules [67, 68, 69].

However, MIDI 1.0 exhibited well-documented limitations, including low resolution (7-bit CC messages), unidirectional communication, and the need for manual mapping, all of which pose significant barriers to users with impairments [70, 71].

The release of MIDI 2.0 in 2020 marked a transformative shift. The new protocol retains backward compatibility but introduces critical enhancements, which will be discussed in Chapter 4 along with their implications for accessibility.

3.2 ADMIs in Music Ensembles and Inclusive Settings

While the design of ADMIs often focuses on the individual user's interaction with the device, the context of the music ensemble and inclusive settings introduces a complex layer of social, therapeutic, and technical interdependence. The integration of ADMIs into ensemble settings, indeed, requires a shift from a purely individualistic focus – centered on overcoming personal physical or cognitive barriers – toward a broader consideration of the performance ecosystem. While the design of ADMIs has historically prioritized ergonomic adaptation for the individual musician, their application in group contexts introduces new dynamics related to social interaction, distributed agency, and the interdependence of performers. In an ensemble setting, an ADMI ceases to be a solitary interface and becomes a node within a broader ecology of "musicking", where the connective functions of music-making facilitate social cohesion across individual differences [72]. The transition from solo performance to ensemble playing requires a shift in design philosophy, moving away from single-user paradigms toward modular, interactive units that foster collaborative musical processes[73]. On the other hand, the use of ADMIs in therapeutic contexts such as music therapy allows them to act as facilitators in the therapist-patient relationship and in the patient's sound identity.

3.2.1 The Social Dynamics of Inclusive Ensembles

The integration of ADMIs into ensembles challenges traditional musical ontologies, specifically regarding what constitutes a musician and how performance is structured. Research by the Performance without Barriers group [72] suggests that music improvisation often serves as the most effective framework for inclusive ensembles. Unlike the rigid hierarchies of traditional Western music, which may demand normative bodily capabilities to execute a fixed score, improvisation allows for "collaborative emergence". In this context, the ADMI allows the musician to engage in real-time creative decision-making and risk-taking, shifting the focus from a "lack" of ability

to the unique sonic contribution of the performer. However, true inclusion in an ensemble goes beyond simply placing a disabled musician alongside non-disabled peers; it requires a reimagining of the performance space to ensure that the technology facilitates genuine dialogue. Embodied listening and wearable interfaces, for example, can contribute to a more felt, shared experience between performers, bridging the gap between digital sound production and physical presence [74] [53].

3.2.2 Facilitated Performance and Interdependent Musicianship

In ensembles involving musicians with complex disabilities, the interaction is seldom solely between the performer and the instrument. Instead, it frequently operates within a framework of facilitated performance, where a facilitator (such as a music therapist, technologist, or fellow musician) plays an active, real-time role in the creative process. Research indicates that in these settings, there is a secondary "hidden interaction cycle" between the performer and the facilitator, distinct from the primary cycle involving the audience [75]. The facilitator does not merely provide technical support but acts as a "gatekeeper" to musical expression, managing technical parameters such as adjusting sensor sensitivity or selecting virtual instruments—to match the performer's immediate capabilities. Dickens et al. [75] identify two distinct interaction cycles in this context:

- Cycle 1: A hidden interaction between the performer and the facilitator (e.g. a technologist adjusting code, sensitivity, or mapping in real-time).
- Cycle 2: The public interaction between the performer, the ensemble, and the audience.

This dynamic parallels what McPherson and Kim describe as "the problem of the second performer", where the introduction of augmented or digital instruments creates a dependency on a technologist who may not be visible on stage but whose agency is critical to the performance [76]. Consequently, the "instrument" in an accessible ensemble is not just the hardware controller, but a distributed system involving the performer, the software, and the facilitator.

This interdependence challenges the traditional Western ontology of the autonomous virtuoso, replacing it with a model of distributed agency where the "instrument" is effectively a system comprising the hardware, the software, the performer, and the facilitator. Successful inclusive ensembles, such as the Able Orchestra[1], demonstrate that this collaborative layering allows for the creation of high-quality artistic work that transcends the "therapeutic" label often applied to disability arts.

[1] Able Orchestra (AO) https://www.inspireculture.org.uk/arts-culture/able-orchestra-ao/

3.2.3 Collaborative Accessible Digital Musical Instruments (CADMIs)

Recent developments have moved beyond adapting single-user instruments toward designing Collaborative Accessible Digital Musical Instruments (CADMIs). These systems are explicitly designed to foster social interaction and shared musical control, addressing the isolation that can occur with headphone-based or single-user ADMIs.

A prominent example is the LoopBoxes project, which utilizes a modular design to support collaborative loop-based music-making in Special Educational Needs (SEN) settings [73]. By physically decoupling the interface into tangible modules that can be shared, the system enforces collaboration, requiring musicians to listen and respond to one another to construct a cohesive musical piece. Similarly, the I-Ork (Inclusive Orchestra) project employs a multidisciplinary approach to create a "friendly musical environment" where timing constraints are relaxed, and event sequences are programmable [77]. This allows musicians with diverse cognitive abilities to contribute rhythmically and melodically without the pressure of normative precision, fostering a sense of shared ownership over the performance.

3.2.4 Improvisation as an Inclusive Framework

Improvisation serves as a critical pedagogical and artistic framework for ADMI ensembles. Unlike the rigid structures of notation-based performance, improvisation prioritizes listening, turn taking, and sonic exploration qualities that are inherently accessible. Research by the Performance without Barriers group emphasizes that improvisation technologies (e.g. VR instruments or sensor based systems) can act as "boundary objects, bridging the gap between musicians of different abilities by focusing on sonic texture and gesture rather than virtuosic physical technique" [78].

Following the principles of Ability-Based Design (ABD), the context of the ensemble fundamentally alters the requirements of the instrument. A device that functions well in a solitary, controlled environment may fail in a live ensemble due to "situational impairments" [79]. For example, stage lighting may interfere with eye tracking ADMIs, or high ambient noise levels from other instruments may render auditory feedback loops, essential for some visually impaired musicians, ineffective.

Furthermore, the concept of performativity in these settings can extend to the very act of accessibility itself. In sign-language-interpreted music, for example, the accessibility provision becomes a performative element, where the interpreter embodies the non-verbal elements of the music (rhythm, pitch, intensity) alongside the performers, creating a unified, multi-modal ensemble experience [80]. Despite the existence of a "performative moment", the instant of decision making in front of an audience requires ADMIs to possess a high degree of reliability and "performative

force".

If an ADMI is too complex or requires extensive setup time between songs, it disrupts the flow of the ensemble, potentially excluding the musician from the "theatrical" dimension of the event. This underscores the need for robust maintenance strategies and communities of practice (such as DMLab) to ensure that custom ADMIs remain functional and adaptable over long periods of use, rather than becoming obsolete after a single project. Morover, the sustainability of these ensembles relies heavily on communities of practice. The longevity of custom ADMIs in group settings is often threatened by "knowledge and skill dependencies" if the specialist technologist leaves, the instrument often falls into disuse. Therefore, sustainable inclusive ensembles require a knowledge sharing ecosystem involving performers, facilitators, and designers to maintain and adapt the technology over time [81].

3.2.5 The Relational and Therapeutic Dimension of Music Accessibility

The integration of ADMIs in inclusive settings is deeply connected to the principles of music therapy, where music is understood not as a "curative pill", but as a primary medium for human relationship. In this perspective, music therapy functions as a relational space that opens communicative channels alternative to verbal language, which is often a significant barrier for individuals with complex disabilities [82] [83]. From a clinical and psychological standpoint, sound is an elective mediator in communication. It is widely recognized in the music therapy literature, most notably in [84] that sound is the first sensory experience to develop, beginning in the womb. This early ontological connection provides sound with a profound "regressive capacity", allowing it to bypass cognitive filters and tap directly into the user's memories, emotions, and "lived sonic experiences" (*gestes musicaux* in french) [85]. Benenzon's theory of the Sound Identity (ISO) is crucial here: every individual possesses a unique internal sound identity that reflects their history and personality [86]. For a person with a disability, whose senses may be more acute or differently tuned with their physical or cognitive abilities, an ADMI acts as a facilitator in the therapeutic relation (musical-sound improvisation) with the user's sound identity. By removing the physical barriers of traditional instruments, these digital tools offer a "sonic palette [82]" or what Gerardo Manarolo [84] poignantly describes as the "musics of the heart" ("musiche del cuore", in italian), allowing the user to project their internal emotional state into a shared space. The accessibility of a digital instrument, therefore, is not just about a technical "performance," but about restoring autonomy, agency, and awareness. In a therapeutic relationship, the ADMI becomes a tool for the person to exercise choice, a fundamental human right that is often diminished by social definition of disability. As Postacchini et al. [83] and Lecourt [87] emphasize, the

therapeutic setting provides a safe playground where music is used as an interactive game to foster self-expression and social cohesion. For the therapist, who understands the underlying sonic dynamics, the ADMI serves as a sophisticated interface to interact with the user's sonic symbols. This creates a bidirectional communicative flow: the instrument facilitates the user's expression while providing the therapist with a clearer window into the user's emotional world. In this context, accessibility is the prerequisite for the "interdependent musicianship" (explained in Section 3.2.2), transforming the act of making music into a powerful vehicle for rehabilitation, well-being, and social inclusion.

3.2.6 Theoretical Framing: Agency, Participation, and Creative Empowerment

The design and evaluation of ADMIs are not merely technical exercises but are deeply rooted in the promotion of musical agency. In the context of artistic research and music therapy, agency is defined as the individual's capacity to act intentionally within a musical system, transforming passive reception into active contribution. This concept is deeply explored by DeNora [88], who posits that music is a technology of the self through which individuals exercise their agency to regulate emotional states and social interactions In the context of ADMIs, agency is realized when the interface allows the user to project their internal ISO into a shared communicative space, effectively restoring the power of choice and action. As noted by Rolvsjord [89], a resource-oriented approach in music therapy emphasizes that the removal of physical barriers through technology allows the "musical self" to emerge, regardless of functional limitations. By shifting the focus from deficit to potential, the ADMI becomes a vehicle for agency, where the user is no longer a patient but a musician with the power to shape sonic environments.

This restoration of agency is the prerequisite for creative participation. According to the framework of Community Music Therapy [90], participation is a relational process that goes beyond simple interaction with a tool. It involves the inclusion of the individual in a "musicking" ecosystem where social and artistic values are co-constructed. The inclusive MIDI controller project of this thesis (explained in details in Part II), by allowing for distributed agency and facilitated performance, enables a level of participation that is not contingent on normative motor skills but on the shared intentionality of the ensemble.

The convergence of these elements leads to Creative Empowerment. In artistic research, empowerment is understood as the process through which individuals gain control over their expressive tools and consequently their artistic identity [91]. For musicians with impairments, empowerment is realized when the technology disappears

into the background, allowing the immediacy of the creative gesture to take precedence. Thus, the interpretation of evaluation results in this thesis must be viewed through this lens: the success of the instrument is measured not only by its usability scores but by the degree to which it fosters a sense of ownership and autonomy in the user's creative life.

3.3 HW/SW Ecosystems for Inclusive Music Education

In this section, we propose a computer-based approach to address the educational needs of pre-school-age children and people with different types of disabilities, both cognitive and physical. In particular, we focus on the potential of button-based interfaces, analyzing the range of actions that a user can perform and a system can consequently detect. After associating a musical meaning to such actions, we propose an accessible and inclusive HW/SW platform capable of combining different peripheral devices to control a music production system, e.g. a synthesizer.

Analyzing the potential of a button interface is valuable for several reasons. First and foremost, buttons are ubiquitous in everyday life, playing a crucial role in numerous contexts, such as calling an elevator or activating a pedestrian traffic light. This familiarity makes them immediately recognizable and eliminates the need for extensive explanation or instruction.

Secondly, their intuitive design is particularly advantageous in accessibility contexts. Buttons require minimal cognitive effort to understand and operate, making them suitable for individuals with cognitive disabilities. Suffice it to say that a specific type of button interface, known as "mushroom push button", is commonly used as an emergency stop or panic switch. Furthermore, simplicity and ease of use make buttons accessible to many people with physical impairments, including those with limited dexterity or strength.

Moreover, narrowing the field to music education, button interfaces are particularly relevant because they align with "traditional" musical controls. Examples include keys on keyboard and wind instruments, effect pedals, and drawknob stops for organ registers. Buttons are also typically available on the user interfaces of media players and synthesizers. These familiar controls not only simplify the transition to digital or adaptive musical tools but also make the learning process more intuitive by building on existing knowledge. This connection to established musical practices highlights the versatility and accessibility of button-based designs, making them an excellent choice for fostering engagement and inclusion in music education settings.

Lastly, buttons are widely available and relatively inexpensive, which contributes to their practicality and feasibility in various applications. Their cost-effectiveness

makes them an ideal choice for projects aiming to create inclusive and affordable solutions, ensuring broader access and usability.

Push-button interfaces have been widely discussed in the scientific literature, exploring this control device's historical, anthropological, and technological implications. To mention but a few works, Plotnick traced the origins of today's push-button society by examining how buttons have been made, distributed, used, rejected, and refashioned throughout history, specifically focusing on the period between 1880 and 1925 [92]. Gaspar *et al.* analyzed the design requirements of buttons and their relations with ergonomics [93]. The relevance of this type of interface is also witnessed by research trying to reconstruct or simulate push actions in alternative ways, such as ad-hoc haptic interfaces [94, 95, 96] or touchless interactions with public displays [97]. Of particular interest to our study, due to its proximity to sound-related aspects, is a research work that investigates the relationship between the signal properties and the perceptual attributes of everyday push-button sounds [98].

3.3.1 Actions and Events Using a Physical Button

The interaction with a single physical button through a computer system involves various types of actions and events, each with its unique characteristics, applications, and implementation details. These actions can be categorized into basic, advanced, and derived actions, reflecting their complexity and potential use cases. Table 1 provides a synoptic overview of the button actions described in this section.

Basic Actions

Basic actions represent the fundamental interactions with a button. The most straightforward action is the *button down* event, which occurs when the button is pressed down, initiating contact. Implementation requires detecting the state change from "up" to "down" as soon as it happens, ensuring low latency and accuracy.

Conversely, the *button up* event occurs when the button is released after being pressed. This event often signals the end of an action started by a *button down* event. It is essential to detect this transition promptly to avoid inconsistencies in the user experience.

A combination of these two events forms the *button click*, a quick press and release. This is widely used for discrete actions such as selecting items in a menu or triggering a one-time function. The implementation must ensure that both the press and release occur within a predefined time interval, avoiding confusion with other actions like holding or multi-clicking.

The *button hold* action occurs when the button is pressed and held for a sustained duration. It is typically employed for continuous actions or mode switching, such as dimming a light or activating a special feature. Implementation requires measuring

Action	Description
Button Down (Press)	The button is pressed down, initiating contact.
Button Up (Release)	The button is released after being pressed.
Button Click	A quick press and release of the button.
Button Hold	The button is pressed and held for a sustained period.
Double Click	Two quick successive presses and releases of the button.
Multi-Click	Three or more successive quick presses and releases.
Hold and Release	The button is held down for a set duration before release.
Long Hold	The button is held for an extended duration (e.g., more than 5 s).
Press with Modifiers	Combining a button press with other inputs (e.g., simultaneous button presses).
Press and Rotate (with Encoders)	Pressing a button integrated with a rotary encoder while turning it.
Pressure-Sensitive Actions	Detecting varying levels of pressure applied to a button.
Sequential Presses	A defined sequence of presses (e.g., press-release-press).

Table 1: Actions and Events Using a Physical Button. The Categories (from Top to Bottom) Refer to Basic Actions, Advanced Actions, and Derived Actions, Respectively.

the duration of the press and triggering the appropriate response only when the hold time exceeds a certain threshold.

Advanced Actions

Advanced actions extend the capabilities of button interaction by introducing more complex patterns or timing-based logic.

A *double click*, for instance, involves two quick successive presses and releases. This action is commonly mapped to alternate functions, such as opening a file instead of selecting it. Detecting a double click requires timing logic to ensure that the interval between clicks falls within a predefined range.

A related but more complex action is the *multi-click*, where three or more quick successive presses are detected. This can unlock advanced features, such as resetting

a device or triggering specific modes. The implementation challenge lies in accurately detecting the intended number of clicks without interference from noise or user input variations.

The *hold and release* action combines a sustained press with a release event. This allows for time-dependent actions, such as activating a special function if the button is held for more than n seconds. Accurate timing measurements and threshold checks are essential for this implementation.

In the previous case, the value of n is usually low, in the order of a few seconds. The *long hold* action is a specific variation where the button is held for an extended duration, often exceeding 5 seconds. This action is reserved for critical functions such as performing a factory reset or enabling a secure mode. Differentiating between a short and long hold requires careful calibration of timing thresholds to prevent accidental activations.

Finally, *press with modifiers* involves combining a button press with other inputs, such as simultaneous presses of additional buttons. This enables complex interactions, such as keyboard shortcuts. Implementation requires monitoring the states of multiple inputs and coordinating their timing.

Derived Actions

Derived actions leverage additional hardware capabilities or sophisticated input patterns to provide enhanced functionality. One example is *press and rotate*, which occurs when a button integrated with a rotary encoder is pressed while being turned. This action is ideal for precise adjustments, such as navigating menus or fine-tuning settings. Synchronizing the signals from the button press and the rotary encoder is crucial for effective implementation.

Another derived action is *pressure-sensitive interaction*, where varying levels of pressure applied to the button are detected. This is particularly useful in contexts requiring dynamic control. However, implementing this action requires specialized hardware capable of detecting pressure levels.

The *sequential presses* action involves a defined sequence of presses, such as press-release-press. This is often used for unlocking specific actions, such as entering a password or toggling modes. Accurate tracking of the input sequence and timing is necessary to differentiate valid patterns from unintended presses.

Across all these actions, certain considerations are universal. First, *debouncing* is critical to mitigate false detections caused by electrical noise or mechanical imperfections in the button. This activity can be performed by either hardware or software components in order to filter out unintended state changes, thus ensuring reliable input recognition.

Second, focusing on user experience, the balance between simplicity and complexity is vital. While basic actions are intuitive and straightforward to implement, understand, and perform, advanced and derived actions provide greater versatility at the cost of additional implementation complexity, user learning, and concentration.

The actions we have described so far focus on the use of a single button. The availability of multiple buttons would enable a wide range of applications. An example is provided by the already-mentioned action called *press with modifier*. Even the interfaces of traditional musical instruments, such as keyboard instruments, can be seen as combinations of buttons: in the case of a piano, the keys are sensitive to activation (with variations in pressure) and release, and the pedals can be considered switch controllers.

Finally, the application context greatly influences the design of these interactions. For instance, accessibility-focused systems benefit from simple and easily distinguishable actions, while musical performance systems require precise timing and responsiveness. By tailoring the implementation to the specific use case, physical button interactions can be optimized for both functionality and usability. The adaptation of the button paradigm to inclusive music education will be addressed in the following section.

3.3.2 Button Interfaces for Inclusive Music Education

The button actions described above, forming the core of user interaction in music production environments, can be employed in educational settings to support music learning. These actions offer a wide range of functionalities that can be adapted to suit the needs of learners. Understanding how these button actions can be used, along with considerations for accessibility and inclusivity, is crucial for ensuring that all users can engage with music technology effectively. In the remainder of this section, we will begin by examining single-button actions and then explore the possibilities enabled by the simultaneous use of multiple buttons.

Single Button

When considering music production, *button down* can trigger musical events, such as playing a note on a traditional instrument or a MIDI keyboard. When a student presses a key, they may hear a corresponding sound, which allows them to experiment with pitch, timbre, and harmony. This basic interaction is an intuitive way for beginners to get started with creating music and experimenting with sounds. In music education, *button down* actions can also be used to trigger pre-recorded sequences or exercises in music software. For example, pressing a button could start a metronome or initiate an exercise designed to practice rhythm or note recognition. For students with physical disabilities, particularly those with limited dexterity, the button press

should be designed to require minimal effort, possibly utilizing larger or softer buttons with customizable sensitivity. This ensures that learners with motor impairments can still engage with the content without feeling overwhelmed by the physical demands of the interaction. Such a consideration can be extended to all button actions described below.

The *button up* action is usually associated with the end or completion of an action, such as releasing a key on a keyboard controller. When applied to sound production, this action is essential for defining the duration of notes (even if a delay effect or controls such as the sustain pedal can prevent the sudden release of sound). In this sense, button release can be used to teach concepts like note length and rhythm. Considering *button up* as the counterpart of an action started by a *button down* event, such an action could also stop the reproduction of an audio file or the synthesis of a note sequence previously started. Concerning accessibility, software or hardware could offer adjustable settings for note release time to assist students with motor impairments, helping users to adhere to a predefined time grid or release the button in a controlled manner.

The *button click* action is used extensively in user interfaces, and, consequently, also in music production and education software, often performed through a mouse button or a keystroke. In this sense, use cases are endless: for example, in a DAW, clicking a button might trigger a play/pause event or allow the user to select an instrument or change a sound preset; in educational applications, such an action could trigger interactive lessons, quizzes, or feedback on musical tasks, such as identifying scales or intervals. But the scenarios described so far do not refer to the musical meaning of the *click* action, rather they showcase educational environments that can be controlled, like almost any software, by keyboard and mouse. Rather, focusing on the musical meaning of a *button click*, this action could be associated with the production of impulsive sounds and rhythmical patterns or the response to a stimulus through a prompt reaction. A sequence of *button clicks* can also be employed as an alternative to *button down*/*button up* or *button hold*: for example, instead of triggering a note attack via *button down* and releasing the sound via *button up*, the user could perform two distinct *button clicks*. In terms of accessibility, buttons should be clearly labeled, with visual or auditory cues to confirm the action, so that students with visual impairments or learning difficulties can easily navigate the interface. The use of *button clicks* as an alternative to *button down*/*button up* could be helpful when a hold action is too demanding.

The *button hold* action involves pressing and holding a button for a longer period. In music production, this can be used, e.g., to sustain a note, much like a sustain pedal on a traditional keyboard, allowing for expressive control over musical performance. In music education, a button hold could be employed to teach students about the concept of sustained notes or to control the duration of a note in an exercise. This

action is particularly beneficial in the context of limited fine motor control, as it may be easier to sustain a button press rather than perform quick taps.

In general terms, the family of Advanced Actions can be invoked in a more advanced training phase. Some actions, such as *double click*, *long hold* and *press with modifiers*, can be physically or cognitively demanding. On the other hand, one of the advantages of adopting simple interfaces such as buttons lies in the possibility to tailor activities based on the student's current abilities and the expected skills; in this sense, advanced actions, if properly calibrated, can represent engaging challenges. Moreover, for users with physical disabilities, there are customization options allowing for slower or alternative input methods such as voice commands.

The *hold and release* action is an advanced type of interaction that can be used to control continuous parameters, such as modulation depth or volume. For instance, holding a button might gradually increase the volume of an audio track until the button is released, returning the volume to its original state. In music education, this kind of interaction could be used to teach students about dynamic changes in music, such as crescendo and decrescendo. Accessibility features should include adjustable duration thresholds for holding the button, ensuring that users with motor impairments can control the action effectively.

As mentioned in Section 3.3.1, derived actions are more specialized interactions that combine multiple input types. Even if potentially powerful in controlling multiple dimensions simultaneously, these actions can be perceived as challenging to perform from a motor perspective (consider scenarios involving precise control of pressure or rotation) and complex to learn and associate from a cognitive perspective. One example is *press and rotate*, where a button is pressed while being rotated. This is commonly used to adjust parameters like volume or filter cutoff in music production. In educational contexts, press-and-rotate controls can help students engage with interactive exercises that involve adjusting multiple parameters simultaneously, such as tuning a virtual instrument or adjusting sound properties in real time.

Pressure-sensitive buttons are another example of derived actions, where the pressure applied to a button affects the outcome. In music production, pressure-sensitive pads are commonly used to control the velocity of notes played on a MIDI controller, allowing for dynamic control over the intensity of sound. This provides musicians with the ability to add expression to their performance. In music education, pressure-sensitive buttons could be used to help students understand concepts like dynamics (soft and loud) and articulation. To ensure accessibility, pressure-sensitive buttons should allow for customization in terms of sensitivity, making it easier for students with different levels of motor control to interact with the system.

In general, sensory limitations can play a role. Users with reduced tactile sensitivity might struggle to distinguish between buttons or confirm if they have been pressed correctly, while those with visual impairments may face difficulties if the buttons or

their states are not clearly visible.

Multiple Buttons

In an accessibility context, requiring the use of multiple buttons for performing actions can present various challenges for users with disabilities. Motor coordination can be a significant issue, as some users may lack the fine motor skills needed to press multiple buttons simultaneously or in quick succession. Those with limited mobility, such as individuals with conditions like cerebral palsy or paralysis, may find it particularly difficult to operate multiple buttons at once. Additionally, reduced finger strength or dexterity can make pressing or holding buttons challenging.

From a cognitive perspective, the complexity of remembering sequences or combinations of button presses can be overwhelming for users with memory impairments or cognitive disabilities. This increased complexity also raises the likelihood of errors, which may lead to frustration or even task abandonment. Environmental factors can exacerbate these difficulties. For instance, buttons requiring simultaneous use may be physically out of reach for some users or may demand excessive or uneven force to operate, adding to the challenge. Usability concerns, such as the potential for fatigue from repetitive or prolonged multi-button actions, are also significant. Precision requirements for pressing multiple buttons correctly can further diminish accessibility and lead to frustration.

The challenges described here, if appropriately adapted under the guidance of an expert, can be transformed into opportunities for learning, rehabilitation, and inclusive music-making. It is essential that the goals are tailored to the individual's abilities and that the tasks do not lead to frustration.

Some advanced actions mentioned above fall under the umbrella of multiple-button availability. For example, we can trace back *press with modifiers* to the combined use of independent buttons. In a traditional setting, playing a chord on a piano can be seen as a press (the root note) with modifiers (the other notes in the chord) action; still, the mentioned elements are independent keys whose role (either the main button or a modifier) can change during the performance. In music education, this approach can be used, e.g., to learn the principles of harmony, but, similarly to other advanced actions, it requires dexterity and can be cognitively challenging.

Sequential presses involve pressing multiple buttons in a specific order to trigger an event. In music production, this could be used to activate a series of effects or sequences, such as arpeggios or drum patterns. In music education, sequential presses can be used in exercises that teach students about musical structure and sequence, such as arranging notes or completing rhythm patterns. Accessibility considerations include allowing for flexible timing between presses and providing feedback to indicate whether the sequence was completed correctly.

Finally, let us mention the scenario where multiple buttons are used by independent players to perform basic, advanced, or even derived actions. This approach can mitigate the usability issues discussed above. Furthermore, collaborative performances foster inclusivity by encouraging social interaction, teamwork, and mutual understanding among participants of diverse abilities and backgrounds. They empower individuals to express themselves, contribute meaningfully, and feel valued within a group.

3.3.3 Relevant Examples

As mentioned above, the interfaces of many "traditional" musical instruments and sound equipment implement forms of interaction that may recall button actions. But, in this section, we want to address computer-based approaches that explicitly rely on the button paradigm. Due to their interface, such tools fall under the wider umbrella of tangible user interfaces (TUIs), which provide innovative ways for users to interact with digital systems through physical objects [99]. In the context of music production and learning, music TUIs enable more intuitive and creative musical experiences [100]. By bridging the gap between tactile interaction and digital sound processing, music TUIs foster a deeper connection between the performer and the music, often with significant benefits for accessibility and education.

One example of a music-focused TUI is the *Kibo* by Kodaly,[2] a MIDI Bluetooth instrument crafted from solid maple wood [101]. The system includes a modular keyboard composed of eight distinct magnetic shapes, each representing a different musical note or control parameter (see Figure 5). Users can rearrange these shapes to create custom musical phrases. Through its dedicated app, the *Kibo* translates these physical interactions into musical output, not only triggering notes but also offering features such as tone adjustments, scale selection, and octave control. This integration of tangible and digital elements makes the *Kibo* particularly effective in music education, especially for young children and individuals with disabilities [102]. The *Kibo* implements several button actions: its eight tangibles are pressure-sensitive, thus supporting *button down*, *button up*, *button click*, *button hold*, and *pressure-sensitive interaction*. Moreover, the board has a knob control to perform *press and rotate* action. All these controls produce standard messages via the MIDI protocol and can be associated with different musical parameters.

The *Skoog* by Playable Technology is a tactile, cube-shaped musical interface designed for accessibility (see Figure 6). Users can press, squeeze, or tap the *Skoog* to produce sound, which is mapped to various instruments or MIDI controls. The device connects to apps that allow users to select scales, keys, and instruments, making it a versatile tool for inclusive music education [103]. The *Skoog* can also be used as

[2] `https://www.kodaly.app/`

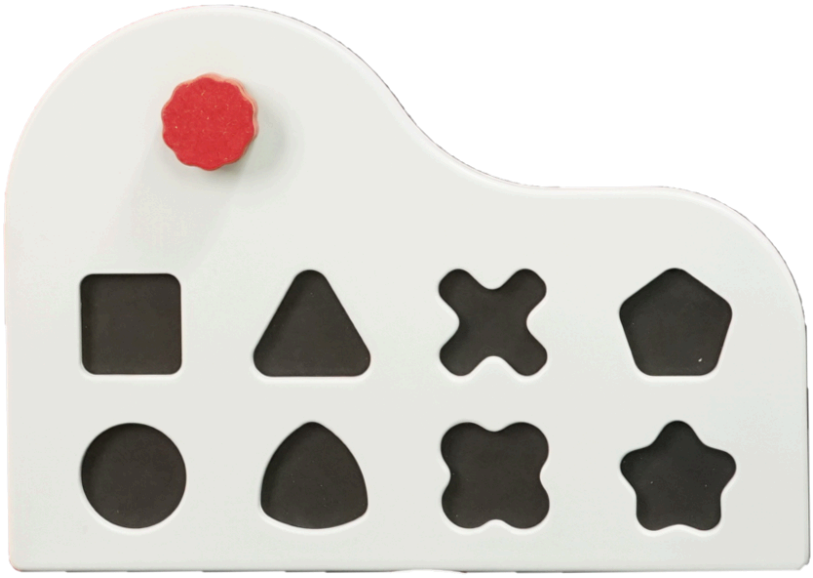

Figure 5: The *Kibo* by Kodaly.

a simple 5-button communicator when paired with an iPad. Its soft, responsive surface makes it particularly suitable for children, individuals with motor impairments, and those new to music-making. The *Skoog* implements *button down*, *button up*, *button click*, and *button hold* actions. Unfortunately, this project has recently been discontinued.[3]

AudioCubes by Percussa [4] are wireless cubes that use light sensors to control sound and music. When cubes are moved, rotated, or placed near each other, they send MIDI or OSC signals to music software, enabling dynamic and visually engaging performances. All the faces are equipped with proximity sensors capable of detecting other objects, hands included; in this way, each *AudioCube* can turn into a multiple-button interface, also sensitive to pressure. This modular system provides an open-ended platform for exploring composition and live performance, encouraging experimentation through tactile interaction.

The *Soundplane Model A* by Madrona Labs [5] is a tactile music interface that combines the expressiveness of a stringed instrument with the capabilities of a MIDI controller. It can detect a wide range of touches on its playing surface, from a light tickle to a very firm press. While it is not a traditional button-based interface, it has been chosen to exemplify the tactile qualities that define TUIs in music.

[3] `https://www.playable.tech/`

[4] `https://www.percussa.com/what-are-audiocubes/`

[5] https://www.madronalabs.com/soundplane

Figure 6: The *Skoog*.

The music-focused TUIs mentioned above demonstrate the potential of tangible interaction to enhance creativity, accessibility, and engagement in music-making. These systems encourage exploration and learning by offering immediate feedback and a hands-on approach to sound manipulation. TUIs break down barriers to music education, making it accessible to people of all ages and abilities. Furthermore, these devices can form an ensemble, operating with other identical or similar tools, electronic equipment, or traditional instruments. Music-making's social and connective functions are fundamental to fostering interaction, integration, and cooperation among learners [104, 105, 78].

This research highlights the versatility of push-button actions, ranging from basic interactions like clicks and holds to more advanced patterns such as sequential presses and pressure-sensitive input. These actions are mapped to diverse musical functions, ensuring that users with varying motor skills and cognitive abilities can actively participate in music-making activities.

Chapter 4

MIDI 2.0 for accessibility

4.1 Why MIDI 2.0

As already mentioned in the previous chapters, MIDI, standing for Musical Instrument Digital Interface, is a well-known technical standard that, despite its decades-long history, is still widely adopted in the field of electronic music generation and musical information description. The MIDI specification describes the communications protocol, the related digital interface, and the electrical connectors designed to support a wide range of electronic music-oriented devices (e.g., musical controllers, synthesizers, sequencers, etc.) [106]. About 12 years after the release of MIDI 1.0 specification, a file format called SMF (Standard MIDI File) was also standardized with the aim of saving and reproducing sequences of MIDI messages and related metadata [107].

A recent milestone in the evolution of this technology is MIDI 2.0 [108], first released in early 2020. MIDI 2.0 represents an extension of MIDI 1.0 rather than a stand-alone specification. In other terms, MIDI 2.0 does not replace MIDI 1.0 but builds on its core principles, architecture, and semantics.

The foundational architecture for MIDI 2.0 expansion is described in the *MIDI Capability Inquiry* (MIDI-CI) specification [109]. MIDI 2.0 compatible devices support bidirectional communication and the mutual exchange of information and profiles. MIDI-CI lets such devices agree to use extended MIDI capabilities beyond those already defined in MIDI 1.0 while preserving backward compatibility. In this way, MIDI systems can embed both MIDI 1.0 and MIDI 2.0 devices so as to support the wide range of musical instruments and SW/HW tools released in the last decades. Morover, MIDI 2.0 introduces critical enhancements, including 32-bit parameter resolution and profile configuration. These innovations support auto-configuration, allowing devices to announce their capabilities and dynamically adapt to host software, a crucial feature for assistive setups, where manual configuration may be impractical. Backward

compatibility is implemented through the use of *System Exclusive*, or simply *SysEx*, messages. This family of MIDI messages is designed to transmit information about specific functions often depending on the product's manufacturer and model. The internal freedom offered by *SysEx* messages made it possible in the past to compensate for some shortcomings of the MIDI 1.0 protocol by standardizing so-called universal *SysEx* messages, accepted by all manufacturers and models interested in adhering to certain standards (e.g., General MIDI). *SysEx* messages can be made up of any number of bytes, potentially very large. *SysEx* messages were already available in MIDI 1.0, thus using them to carry MIDI 2.0 communication is a very simple solution to guarantee backward compatibility. One of the critical issues with *SysEx* messages is that their low level of standardization combined with their typically high number of bytes make them difficult for the user to understand.

Recent literature has begun to explore MIDI 2.0's implications for inclusivity. For instance, MASSIG (Music Accessibility Standard Special Interest Group), coordinated by the MIDI Association, brings together stakeholders from industry and disability advocacy to ensure that MIDI 2.0 remains accessibility-conscious [110]. Projects such as Sound Without Sight[1] have further emphasized the need for adaptive interfaces that support blind and partially sighted musicians. Despite its promise, real-world implementations of MIDI 2.0 in accessible instruments remain limited. Some commercial DAWs have begun to experiment with auto-configuration features, but very few systems actively integrate MIDI 2.0 property exchange for the purpose of inclusive interaction [15]. Some recent research efforts have begun to bridge this gap by demonstrating real-time adaptation to individual user needs, but they often remain experimental or hardware-specific [57, 110].

4.2 A Web-Based MIDI 2.0 Monitor

A category of software tools known as MIDI monitors aims to let users view, analyze, and test MIDI data. Cook [111] describes it as a simple terminal program that displays MIDI messages coming into your system in words rather than numbers. When inserted into a MIDI layout, the monitor displays incoming and outgoing MIDI messages in real time.

The goals and applicability fields of this category of software are manifold. First, a MIDI monitor can help diagnose issues with MIDI devices or connections, which is particularly important for musicians who are not experts with technologies. Software and hardware designers can benefit from MIDI monitoring, too, in testing MIDI drivers and applications. Finally, MIDI monitors can play a fundamental role in educational activities dealing with MIDI since they show the byte values exchanged

[1] `https://soundwithoutsight.org/`

in MIDI communication at a relatively low level of abstraction.[2] Available MIDI monitors can be stand-alone products (e.g., the tools by Morningstar, Morson, and Snoize) or modules integrated into more general software (e.g., Logic Pro, MIDI-OX, Steinberg Nuendo).

This section aims to describe a publicly available MIDI monitor designed and implemented through the Web MIDI API. Although the tool can be profitably used to analyze MIDI 1.0 communication, the major novelty compared to already existing alternatives is the attention paid to interpreting the various parts that constitute MIDI 2.0 messages.

In order to make our MIDI monitor easily available and updatable, we chose to implement it in the form of a web application. This category of software tools refers to applications accessed via a web browser over a network and developed using browser-supported languages. For their execution, web applications depend on web browsers [112].

The *Web MIDI Monitor* was developed using only client-side formats and languages for web programming. In particular, the platform combines HTML and CSS for static content and its presentation, respectively. In addition, its active behavior is governed by JavaScript, a well-known client-side programming language. Since no server-side language is implied, the *Web MIDI Monitor* can operate also in local mode on the user's device.

Concerning the JavaScript code, one project is worth specific attention. The management of the connections and the communication with the external MIDI system was realized through the Web MIDI API, a W3C[3] Audio group initiative that is currently striving to become an officially recognized standard. At the moment of writing, the specification is at the stage of Editor's Draft, i.e. a work-in-progress document allowing the group to iterate internally on its content for consideration [113]. Drafts have not received a formal review and are not endorsed by W3C. Even if not recognized as a standard so far, nowadays the Web MIDI API is well supported by most web browsers.[4] In this sense, a relevant exception is still represented by Apple Safari.

As reported in [113], this specification describes an API for MIDI-protocol support which enables web applications to enumerate and select MIDI I/O devices on the client system and send and receive MIDI messages. By providing access to a wide range of musical instruments, MIDI devices, and compatible software tools, the Web MIDI API enables both music and non-music MIDI applications. Concerning the former aspect, the scientific literature reports research works dealing with its direct use

[2]The MIDI protocol is agnostic with respect to the transport layer, thus a MIDI monitor shows the messages ignoring low-level aspects such as the start and stop bit required by the electrical specification.

[3]World Wide Web Consortium, `https://www.w3.org/`

[4]Source: `https://caniuse.com/midi`

[114, 115, 116], higher-level libraries based on it [117, 118], and experiments mixing it with the Web Audio API, another JavaScript interface developed by the W3C Audio Group [119]. On the other side, the scientific literature has also explored applications based on music concepts but addressing extra-musical aspects, e.g. the development of soft skills, problem-solving attitude, and computational thinking abilities [120, 121].

4.2.1 Basic Functions

In this section, we will describe the software tool developed to perform MIDI monitoring via the web. Such a platform is called *Web MIDI Monitor*[5].

The *Web MIDI Monitor* aims to display in real-time the MIDI messages arriving on all input interfaces recognized in the client system. Since multiple input sources may be available, the message's input port is also displayed.

As an example, a monitoring session is shown in Figure 7. The list includes an *All Notes Off* (corresponding to a specific *Control Change*), a *Program Change*, a *Note-On*, and a *Note-Off* message. In accordance with the expected functions of a MIDI monitor, the tool presents not only the sequence of status and data bytes (in binary, decimal, and hexadecimal format) but also a human-readable interpretation of data. Flags and meaningful aggregations of bits within a byte are highlighted (e.g., the most significant bit to distinguish between a status and a data byte, or the low nibble in the status byte of channel messages to identify the current channel). When possible, numeric information is matched against MIDI tables to offer a straightforward interpretation to the user; in Figure 7, this is the case of the CC number that identifies the *All Notes Off* message within the *Control Change* family, the name of the instrument for *Program Change*, and the note pitch translated into Scientific pitch notation for *Note-On* and *Note-Off*.

Another goal when parsing MIDI information is to reconstruct values that span across multiple bytes. It is the case of the *Pitch Bend Change* message, where 2^{14} detuning levels are expressed on 2 data bytes and remapped onto the range $[-8192, +8191]$, or the scenario of MIDI Universal IDs introduced in MIDI 2.0, that are pseudo-random 28-bit values represented on 4 data bytes. This approach can be further extended to the visualization of values that span across multiple MIDI messages, as in the case of *MIDI time code* that splits an SMPTE timecode into a series of 8 messages.

4.2.2 Advanced Features

The features listed above are present in most MIDI monitors. A novelty aspect of our proposal is the use of web technologies, but, apart from this, there are other

[5] *Web MIDI Monitor*is publicly available at `https://midimonitor.lim.di.unimi.it/`

Figure 7: Example of a MIDI 1.0 message list.

characteristics that are worth underlining.

The main motivation of the project is to provide the user with better support to understand MIDI 2.0 messages. MIDI 2.0 introduces the so-called MIDI-CI communication that relies on suitably-formatted universal *SysEx* messages. MIDI 2.0 specification defines how to interpret a part of the data-byte sequence that, in a manufacturer's *SysEx* message, would be completely free and depending on implementation choices. A notable example is the unique addresses of the sender and recipient device, which must be reconstructed by combining 4 bytes of data each; another case is the value of MIDI profiles expressed in JSON format, which therefore spans across numerous bytes to be interpreted as text characters. In the mentioned scenarios, the mere list of bytes that constitute the *SysEx* message, as shown by generic MIDI 1.0-compatible monitors, would be hardly readable by the user.

The general form of MIDI-CI messages, reported in hexadecimal values, is:
`F0 7E cc 0D xx 01 ss ss ss ss dd dd dd dd [...] F7`

Analyzing the message structure byte by byte:

- `F0` represents the *Start of Exclusive* message, as defined by MIDI 1.0 specification;
- `7E` identifies any non-real time universal *SysEx*, once again in accordance with

MIDI 1.0;

- `cc` is the channel identifier. Value `7F` is reserved to identify the whole MIDI port, namely all channels;
- `0D` is the value of the *Sub-ID #1* field for a generic MIDI-CI message. *Sub-ID #1* and the following *Sub-ID #2* work together to specify the function and sub-function of a universal *SysEx*.[6] This byte is what turns a generic non-real time universal *SysEx*, as already defined in MIDI 1.0, into a MIDI-CI message for MIDI 2.0;
- `xx` is a placeholder for the *Sub-ID #2* part that determines a more specific function with respect to *Sub-ID #1*;
- the following byte provides the MIDI-CI version, currently set to 1;
- the subsequent 4 + 4 bytes contain the MIDI universal IDs (MUIDs) of the source and destination device respectively;
- the variable-length part represents the message's payload;
- finally, `F7` stands for *End of Exclusive*.

Please note that the described structure is far more articulated than in the case of typical MIDI 1.0 messages, whose length ranges from 1 to 3 bytes (with the obvious exception of *SysEx* messages). Moreover, such a structure ignores the content of the payload, which could be further decomposed into smaller information units. Taking into account complexity and making MIDI 2.0 messages readable are the main aspects that motivated the release of a new monitoring tool aware of the MIDI 2.0 extension. We presented the generic MIDI-CI message structure in detail because this internal division guided us in designing MIDI 2.0 visualization.

Minor features include the ability to filter incoming messages by input interface, include/exclude single messages or message families from the monitoring, export the received data in Comma-Separated Values (CSV) format, and set the color palette to improve visualization.

4.2.3 Using the Monitor

In order to see the *Web MIDI Monitor* in action, it is worth clarifying some operating aspects.

[6] The list of MIDI 1.0 supported values is available at `https://www.midi.org/specifications-old/item/table-4-universal-system-exclusive-messages`.

First, the web browser has to be compatible with the Web MIDI API. In this sense, the Monitor is supported also by mobile devices, provided that they have a suitable browser installed.

Secondly, the web tool must be enabled to receive MIDI messages, which means that: i) an upstream part of the MIDI system is required to generate messages, and ii) the client where the application is running must have at least one MIDI input port. The MIDI system to monitor can be a full physical, virtual, or mixed one. The MIDI layout can include both MIDI 1.0 and MIDI 2.0 devices.

Finally, in order to allow the management of *SysEx* messages by the Web MIDI API, the user is required to explicitly grant access to local MIDI devices. Usually, the browser asks for such permission via a dialog window.

To see the *Web MIDI Monitor* in action, it is possible to introduce it into a physical MIDI setup by making the host PC talk with the other MIDI devices in the setup. The simplest way to achieve such a mixed physical/virtual MIDI system is to attach a MIDI controller to the computer via a USB cable, provided that such a connection is supported by the MIDI device, and employ the USB-MIDI transport protocol. A more articulated scenario relies on the use of an external MIDI computer interface that, in addition to IN/OUT ports for traditional 5-pin DIN cables, can deliver MIDI messages via USB. For example, the application was tested by attaching a *Roland PC200* master keyboard to an *M-AUDIO Fast Track Pro* interface, in turn, connected to the computer via USB (see Figure 8). The result of the test session shown in the figure includes a *Note-On* and a *Note-Off* 3-byte message surrounded and interleaved by a number of 1-byte *Active Sensing* messages automatically produced by the *Fast Track Pro* to keep the communication channel alive.

When no dedicated hardware is available, the simplest way to enjoy the potential of the *Web MIDI Monitor* is to launch a software MIDI controller and attach it to a virtual MIDI port, which, in turn, is seen as a virtual input by the browser app. Unfortunately, this approach is unlikely effective to send (and, consequently, track) *SysEx* messages. Rather, this scenario can be emulated using a MIDI sequencer integrating a message editor, so as to enter and send any kind of MIDI commands, including MIDI 2.0 messages. Another option is to produce ad-hoc MIDI messages via a dedicated application, such as the *MIDI Sysex Editor*[7].

Since, at the moment of writing, few hardware devices support MIDI 2.0 and the *Web MIDI Monitor* was specifically designed for such an extension, the generation and monitoring of MIDI 2.0 messages deserve particular attention. To this end, we have installed and configured two MIDI 2.0 software products designed to work together, namely the *Bome MIDI-CI Initiator* and the *Bome MIDI-CI Responder* (see Figure

[7]The *MIDI Sysex Editor*is available at `https://midi.toys/`

Figure 8: Testing the *Web MIDI Monitor* with a physical MIDI setup.

9). In this way, we have simulated a typical MIDI 2.0 information exchange, made of:

- a *Discovery* inquiry by the initiator;
- a *Reply to Discovery* inquiry by the responder;
- an inquiry of *Property Data Exchange Capabilities* by the initiator;
- the *Reply to Property Data Exchange Capabilities* by the responder;
- a number of *Get Property Data* inquiries by the initiator and the corresponding *Reply to Get Property Data* messages by the responder.

The interface of the *Web MIDI Monitor* (see Figure 10) shows the alternating messages sent by the two devices on their output ports. Each port is characterized by a different color, shown on the left side of the message. The structure of each message is interpreted and clearly presented to the user. An example is provided by the 4+4 bytes dedicated to the sender's and receiver's MUIDs.

4.2.4 Discussion

The main advantage of implementing a MIDI monitor as a web tool is the ease of embedding it into a (potentially complex) MIDI system. MIDI communication does

Figure 9: The *Bome MIDI-CI Initiator* and the *Bome MIDI-CI Responder*.

not strictly require the 5-pin DIN connectors standardized by the MIDI 1.0 specification, which are still present in most MIDI musical instruments but are available on standard PCs only through ad hoc interfaces (e.g., joystick ports on old audio cards and external audio peripherals). Luckily, current devices can receive and send MIDI data using recent standard ports and formats through MIDI over USB, Bluetooth, and Ethernet.

Another benefit of a web-based solution is the intrinsic extensibility, with updates made available to users on a rolling base. In the case of MIDI 2.0, which is a quite recent standard extension, the possibility to refine message visualization and management is fundamental, also depending on the needs of developers and the release of new products. For example, the only keyboard controller on the market that claims to be "MIDI 2.0 ready", namely the Roland A-88MKII MIDI Keyboard Controller, never mentions MIDI 2.0 in its operating manual. In the climate of uncertainty that characterizes a moment in which the standard is not yet considered mature by the market, the possibility of quickly updating the MIDI monitor is a potential success factor.

The Web MIDI API is a commonly accepted JavaScript interface under development in the W3C context. On one side, this constitutes a clear advantage, since the management of MIDI messages and the connection with external devices can occur at

Figure 10: Monitoring MIDI 2.0 message exchange through the *Web MIDI Monitor*.

a higher level of abstraction. On the other side, there are some potential drawbacks that are worth remarking on. First, the client computer has to be equipped with one of the compatible browsers, even if the support of this API is getting more and more extensive. Secondly, the management of some features typical of MIDI communication is delegated to the design choices of the Web MIDI API. Examples include how the running status and the occurrence of real-time messages in the middle of other messages are handled. Finally, the communication delay introduced by putting a MIDI monitor in the middle of a MIDI chain is a minor problem in educational and offline debugging activities, but it could impact, e.g., a live-performance scenario. A deeper insight into the Web MIDI API would be beyond the scope of this paper. For a critical discussion about its pros and cons, please refer to [122].

4.3 The importance of MIDI 2.0 in accessibility

The most straightforward use of property exchange and profiles for accessibility is the automatic adaptation of user interfaces, a new feature of MIDI 2.0. The official specifications describe a minimal implementation with a computer running a DAW and a MIDI-compatible pedal [109] that, after being connected to the system, is automatically detected and impacts the graphical user interface of the DAW. This example can be adapted to an inclusive scenario, where MIDI applications and devices query each other to find out what inclusive parameters are supported and configurable [110].

When a musician connects an accessible MIDI controller (e.g., a single-button device, a specific set of buttons, a breath controller, or an eye tracker), the software automatically recognizes its features and adjusts accordingly. This function removes

the barrier of having to set up MIDI mappings manually, which can be challenging for users with motor or cognitive impairments. If a musician uses a simplified input device, the DAW can provide an alternative interface, e.g., with larger buttons, visual cues, or spoken feedback. For blind musicians, the DAW might activate screen reader support or haptic feedback based on the detected controller. The cognitive load can be further reduced by suggesting mappings based on common accessibility presets.

4.3.1 Exchange of MIDI-CI Messages

The following case illustrates the exchange of MIDI-CI messages between a computer running a DAW, playing the role of transaction initiator, and an accessible controller with a 2-axis joystick and a push button, with the role of transaction responder. Provided that both devices are capable of MIDI 2.0 communication, the following exchange of messages occurs:

1. A complete MIDI-CI *Discovery Transaction*, in order to exchange key information between the DAW and the controller (e.g., their *MIDI Universal ID* (MUIDs));
2. A *Property Exchange Capabilities* inquiry sent by the software to request details of *Property Exchange Support* from the controller;
3. A *Reply to Property Exchange Capabilities* message returned by the controller;
4. A *Get Property Data* inquiry sent by the software to request the resource list from the controller;
5. A *Reply to Get Property Data* return message by the controller containing the available resources, i.e., information on the device and the list of all its components, namely two axes and a button;
6. A new *Get Property Data* inquiry sent by the software to obtain some general information about the controller, such as the manufacturer id and name, the family id and name, the model id and name, and the version id and name;
7. A *Reply to Get Property Data* return message carrying the required device information
8. Another *Get Property Data* inquiry sent by the software focusing on the list of controls in the controller;
9. A *Reply to Get Property Data* return message describing single controls (i.e., *X axis*, *Y axis*, and *Button*) and their current values.

MIDI-CI implements the subscribe function, a mechanism of communication between devices that enables them to dynamically discover and negotiate their capabilities. In the scenario described above, the DAW can subscribe to the specific parameters of the controller and receive updates when they change. In this way, the DAW can read these parameters and their ranges, either representing them on the screen or by speech synthesis. The latter behavior, for instance, makes parameter values accessible to people with visual impairments.

Profile Exchange can also facilitate customized interaction models, such as a single button triggering chords or arpeggios instead of single notes, a joystick translating movement into pitch bend or volume control, or a pressure-sensitive switch enabling expressive dynamics for musicians with limited dexterity.

Considering a system with multiple controllers, musicians who use many adaptive devices simultaneously (e.g., foot pedals and sip-and-puff tools) can benefit from Profile Exchange, ensuring that all devices work together seamlessly. The software can even suggest layered control strategies, such as using one device for note input and another for modulation.