2021 IEEE 7th World Forum on Internet of Things (WF-IoT) | 978-1-6654-4431-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/WF-IoT51360.2021.9595417

Smart Musical Instruments preset sharing: an ontology-based data access approach

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

Paolo Bouquet
Dept. Information Engineering and Computer Science
University of Trento
Trento, Italy
paolo.bouquet@unitn.it

***Abstract*—Interoperability represents an important aspect in research dealing with the emerging class of smart musical instruments (SMIs). To date, no interoperable file format for the exchange of content produced by heterogeneous SMIs has been defined yet. This paper proposes a solution to the issue of sharing presets among heterogeneous SMIs, which are used to configure an SMI. The heterogeneity of SMIs may come from the type, structure and implementation of the SMI's embedded system, its sound engine and sensor interface. The presented solution is based on the "ontology-based data access" paradigm and leverages the existing Smart Musical Instruments Ontology. This approach allows one to share presets between heterogeneous SMIs by mapping information about the configuration of an instrument to the concepts of the ontology. Thanks to this approach, SMIs developers can implement programs that convert proprietary formats for the configuration of the instrument into a common format for SMIs, and vice versa. We present the general architecture and workflow of this approach, and we describe an implementation for it which involves the sharing of presets among two heterogeneous smart guitars.**

***Index Terms*—Internet of Musical Things, smart instruments**

I. Introduction

The Smart musical instruments (SMIs) are an emerging class of musical interfaces that encompass sensors, actuators, embedded intelligence, and wireless connectivity to local networks and to the Internet [1]. The central component of an SMI is the embedded system on top of which the intelligent applications run, i.e., a single board computer and a dedicated operating system for audio processing and networking tasks. The sensors composing the sensor interface of an SMI may be used by the player to modulate via gestures the parameters of the internal sound engine of the instrument. Examples of this category of instruments are the smart cajón prototype described in [2] or the Sensus Smart Guitar developed by Elk [3]. SMIs are instances of the so-called Musical Things within the "Internet of Musical Things" (IoMusT) paradigm [4], an extension of the Internet of Things to the musical domain. Within this paradigm, heterogeneous SMIs can exchange content with each other leveraging application and services built on top of the connectivity infrastructure.

Interoperability is the way in which heterogeneous systems talk to each other and exchange information in a meaningful way. An important aspect in SMIs research is that of defining an interoperable file format for the exchange of content produced by heterogeneous SMIs. This topic has been preliminary addressed in [5]. The authors investigated the requirements for the design of a format specific to SMIs, but that at the same time could enable interoperability with other devices. They concluded that the existing standardized formats that are closest to meet the identified requirements are the IEEE 1599 [6] and the IM AF (MPEG-A: Interactive Music Application Format) [7] and that both formats need to be extended with additional features.

In an effort to address an extension of those formats and foster interoperability across heterogeneous SMIs, the Smart Musical Instruments Ontology was proposed in [8]. Ontologies are widely considered in the Internet of Things as a suitable formal tool for sophisticated data access [9]. They are formal representations of a domain of interest, expressed in terms of objects, concepts, links between objects and relationships between concepts. Ontologies are particularly useful because they provide a shared vision of the structure of the domain of interest, and in particular this understanding can be shared both among people and software agents.

A paradigm exploiting ontologies to achieve data interoperability is the "ontology-based data access" (OBDA) [10]. An OBDA system consists of three layers: i) the ontology layer, i.e., the representation of the conceptual domain; ii) the data sources, which are described through schemes and related information; iii) the mapping layer, i.e., the correspondence between the data sources and the concepts and relations of the ontology. In OBDA systems, an ontology is used as a high-level, conceptual view over heterogeneous data repositories, which enables users to access data without the understanding of the data sources, the relation between them, or the encoding of the data. The user can simply pose queries over the high-level conceptual view defined by the ontology and via the mappings these queries are translated into queries directly interfacing with the data sources. Noticeable examples of OBDA systems are MASTRO [11] and Ontop [12].

This paper proposes a solution to the issue of sharing presets among heterogeneous SMIs, which are used to configure an SMI. The heterogeneity of SMIs may come from the type, structure and implementation of the SMI's embedded system, its sound engine and sensor interface. The specific case tackled in this study is that of SMIs of the same kind (e.g., a smart guitar), having the same or a similar sensor interface (e.g., composed by a same number and type of sensors), but with different systems to implement the sound engine and to exploit the data coming from the sensor interface (e.g., with different operating systems or the hardware architecture of the single board computer). The goal is to use a common format for sharing presets able to configure such different SMIs in the same way so that they can generate the same sonic output. Our solution relies on the OBDA paradigm, and is based on the Smart Musical Instruments Ontology described in [8], which thus far has not been integrated in a file format for SMIs.

II. SMART MUSICAL INSTRUMENTS

This section provides a high-level view of the architecture of a smart musical instrument and a description of the smart musical instruments ontology. The provided details are in relation to the task of preset sharing tackled in this study. Therefore, the focus is not placed on other aspects related to the intelligent capabilities of SMIs such as context-awareness and proactivity.

A. Architecture of a smart musical instrument

An SMI is an IoT device composed by various components. The most important components in relation to the present study are the embedded system, the sensor interface and the sound engine (see Fig. II-A). The embedded system consists mainly of a hardware platform, which includes inputs and outputs for audio and sensors, as well as an operating system dedicated to real-time audio and sensors signal processing. Most of existing SMIs prototypes have been built across two embedded systems, the Bela board [13] and Elk Audio OS [14]. Bela consists of a cape for the BeagleBone Black platform and a Xenomai-based operating system, which supports audio processing software such as Pure Data and Supercollider. Elk Audio OS is an operating system for high-quality processing of audio and sensors that is supported for various hardware platforms, including the Raspberry Pi. It is based on the Xenomai Linux extension and on an architecture supporting audio processing via commercial and open source audio plugins in various formats (e.g., VST3, LV2, RE).

The sensor interface is the instrument's component responsible for the tracking of performative gestures conducted by the player. A sensor interface may consist of different sensors, such as a pressure sensor tracking the player's fingers pressure onto a part of the instrument, or an inertial measurement unit tracking the acceleration and tridimensional position of the instrument.

The sound engine of an SMI is responsible for the generation of the instrument's digital sounds. It parallels a typical digital audio workstation consisting of various components, including tracks and a mixer. For instance, a component can process the sounds detected by a microphone by applying digital audio effects to it; a component can trigger sound samples thanks to a sampler; a component can generate sounds resulting from the control of synthesizers and drum machines; a component can play back different backing tracks.

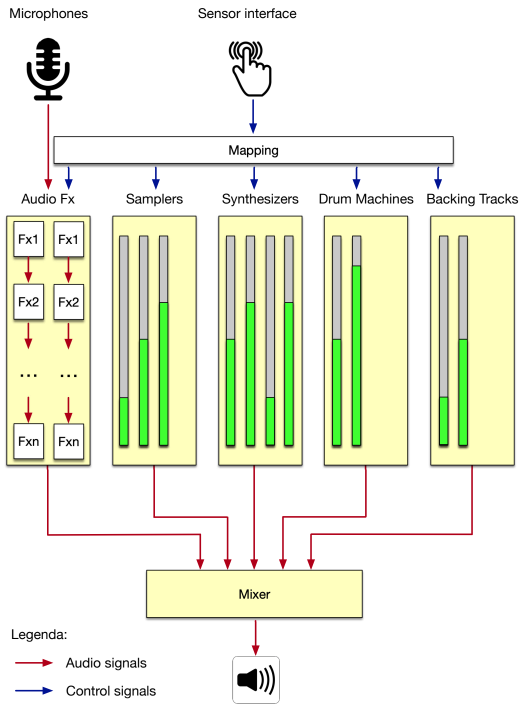


Fig. 1. A schematic representation of the sound engine of a smart musical instrument.

The parameters of each of these components of the sound engine can be modulated by the sensors present in the sensor interface, by means of a set of mapping rules [15]. Bela and Elk Audio OS implement in radically different ways the sound engine, the sensor acquisition and processing, and the mapping between the digitized sensor values and the sound engine parameters. These are some of the aspects that cause heterogeneity among SMIs.

B. The Smart Musical Instruments Ontology

The Smart Musical Instruments Ontology[1] was specifically conceived to address the interoperability issues of heterogeneous SMIs exchanging information between each other [8]. The ontology models the basic concepts of the hardware and software components of an SMI, and relates to several existing ontologies. These includes the SOSA Ontology for the representation of sensors and actuators [16], the Audio Effects Ontology dealing with the description of digital audio effects [17], the Music Ontology that deals with the description of the music value-chain from production to consumption [18], the Studio Ontology for representing the domain of technical workflows occurring in music production [19], and the IoMusT Ontology for the representation Musical Things and IoMusT ecosystems [20].

In more detail, the requirements for the ontology were the following. The SMI Ontology should be able to: *i)* represent the concept of SMIs as an instance of Musical Things, including its type, characteristics (including the number and type of

[1]https://w3id.org/smi#

inputs and outputs), the structure of its sound engine, and the geographical position; *ii)* represent the concept of application and service related to an SMIs, including its purpose, level of interactivity, type, and user; *iii)* describe attributes of the music at a given time, including low-level and high-level features. A practical use of the ontology in an applicative context has been recently reported in [21] for the case of the creation of a database of SMIs.

III. Preset Sharing Architecture

In this section, we describe the workflow and the general architecture of the OBDA-based mechanism for preset sharing across heterogeneous SMIs. The first step of the preset sharing workflow consists of the creation of the preset to be shared. This is achieved automatically by an SMI, which upon an action of the player saves a format specific to that SMI, which contains some configuration files that describe the structure of the sound engine (e.g., number and type of tracks), the components present in each track of the sound engine (e.g., the chain of the digital audio effects), the sensor acquisition and processing mechanism (e.g., sampling rate, resolution, thresholding, range mapping), and the mapping between sensors and the sound engine parameters. This instrument-specific format is then automatically encoded into a general exchange format for SMIs, by leveraging a translation process that implements the mapping between the information contained in the files and the ontology concepts.

Once the preset has been created in the common format for SMIs, the player can share it by uploading it onto an online repository. This process can be performed directly by the SMI thanks to its wireless connectivity feature. A second player, using another model of the SMI, can then download the preset from the repository onto his/her instrument. Subsequently, the preset is decoded from the common SMI format into the instrument-specific format, by leveraging the mapping process from the ontology concepts to the information needed to configure the instrument. As a result the instrument automatically configures itself. Figure III illustrates this whole workflow for the case of preset sharing across two heterogeneous smart guitars. Notably, the encoding and decoding processes can be performed not only directly by an SMI, but also by a server via cloud computing techniques.

In more detail, the steps for passing from a preset encoded in an instrument-specific format to the common format for SMIs include the following. First the instrument-specific format is parsed in order to extract the information relevant for the description of the preset. The extracted information is then mapped to the ontology concepts and as a result a new file is created which is formatted in a common format for SMIs. Vice versa, to pass from the common format for SMIs to a preset encoded in an instrument-specific format the following steps need to be implemented. First the common SMI format is decoded following the ontology concepts in order to extract the information relevant for the configuration of the receiving instrument. The extracted information is then mapped to the fields used by the configuration system of the instrument (e.g., sound engine effects chain, sensor interface) and as a result a new preset file is created, which is formatted according to the format specific to the receiving instrument.

In these encoding and decoding processes, both the sending and the receiving instrument are fully aware of the mappings of the concepts of the ontology. Moreover, it is worth noticing, that not all concepts of the ontology are actually needed to represent the configuration of the sending and receiving instrument for the specific aim of creating/utilizing a preset. For instance, a certain amount of information can be assumed to be already completely available for the algorithms of the receiving instrument at the moment of configuring it as well as of creating the necessary files (e.g., the PD sound engine file). An example is represented by the fact that the sampling rate of the sending instrument and that of the receiving instrument may be different. Such an information is not important to be represented in a preset file. A similar discussion holds for other parameters such as the sampling rate of the analog sensors, which may be different from the original and receiving instrument. We aim at only describing the information strictly needed for the preset, not the whole configuration of the instrument in all its low-level details. As a consequence the proposed method assumes that the receiving instrument already provides the extra information not included in the preset representation deriving from the ontology mappings.

Furthermore, the method is not aiming to replicate exactly each of the possible effect implementations composing the effect chain of a sound engine. What is important is the type of effect utilized (e.g., a reverb). Indeed, in principle it is not possible to assume that a specific effect implementation is available on the receiving instrument (e.g., there might be a different implementation of a reverb effect algorithm). In a practical case, an algorithm would first check if that specific implementation is available on the receiving instrument, and if it is not other automatic mechanisms could be set in place such as asking the user to download the missing audio effect from a repository.

IV. Case Study

In order to demonstrate the usefulness and feasibility of our OBDA approach to preset sharing we report on a real world application in which it has been experimented. Two smart guitars with two different sound engines were created. The first relied on the Bela platform and implemented the sound engine in Pure Data. The second was based on the Elk Audio OS running on a Raspberry Pi 4 in conjunction with a dedicated audio shield and implemented the sound engine according to the operating system tools, which are based on audio plugins.

Both guitars featured wireless connectivity capabilities via Wi-Fi. A local PC acted as a repository server to host the presets in the common SMI format. Both guitars and server were locally connected via a Wi-Fi router. To mimic the user's process of preset saving and uploading/downloading to/from the server, we used a smartphone app wirelessly connected to each guitar. The app consisted of a few buttons allowing the

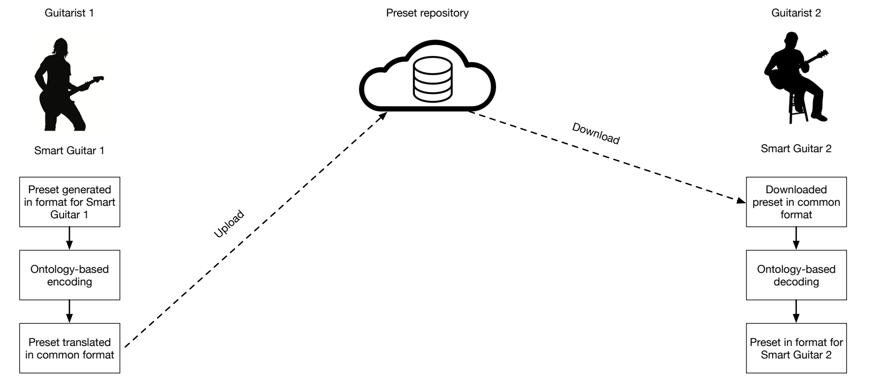

Guitarist 1
Preset repository
Guitarist 2
Smart Guitar 1
Smart Guitar 2
Upload
Download
Preset generated in format for Smart Guitar 1
Ontology-based encoding
Preset translated in common format
Downloaded preset in common format
Ontology-based decoding
Preset in format for Smart Guitar 2

Fig. 2. The workflow for smart guitar preset sharing, from the generation of a preset created by a guitarist for a certain smart guitar model, to its uploading onto an online platform, to the downloading and automatic configuration of a smart guitar of a different model.

users to issue to the respective guitars the control commands of saving, sharing and downloading the preset. The app, built using the TouchOSC framework, leveraged the Open Sound Control protocol for communicating with the instruments.

To illustrate the proposed approach we show a minimal setup of the instruments, where both smart guitars encompassed a simple sensor interface composed by a single pressure sensor and involved a sound engine with a single track consisting of a delay with feedback digital audio effect. The pressure sensor was mapped to the feedback parameter of the delay effect. In the following step-by-step example we focus on the scenario in which the powered-by-Elk smart guitar shares its preset with the powered-by-Bela smart guitar. All software responsible for the processes involved in the preset sharing workflow has been coded in python (leveraging the RDFLib library). The source code is available online[2].

Step 1: preset saving in the powered-by-Elk smart guitar format. The first step consists of saving the configuration of the instrument according to its instrument-specific format. To configure an instrument, the Elk Audio OS provides a JSON configuration file for the sound engine (see listing 1), a JSON configuration file for the sensor engine (see listing 2), and a JSON describing the mappings between sensors and sound parameters (see listing 3). Upon the saving request, these files are merged together in a single preset file.

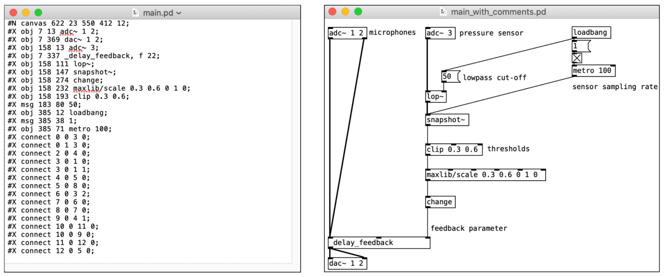
Step 2: conversion to the common SMI format. The preset file generated by the powered-by-Elk smart guitar is then converted into a corresponding file formatted with the common SMI format. This format has been defined as a Turtle file, one of the most common RDF formats used for the Semantic Web. Firstly, the preset file is decoded into the composing configurations files (see listings 1, 2, 3). Secondly, each file is parsed with the aim of extracting the fields relevant to the generation of the common format, and subsequently the mapping between such fields and the concepts of the ontology is performed. Note that not all fields have a general relevance, therefore they can be ignored and excluded from the conversion which by design choice is kept as minimal as possible (e.g., the sample rate of the sound engine is an information that can be safely excluded as the receiving instrument may adopt its own sample rate to work). As a result of this conversion process, the Turtle file is generated (see listing 4) and automatically uploaded to the server.

Step 3: conversion to the powered-by-Bela smart guitar format. After the download of the Turtle file into the powered-by-Bela smart guitar, the process of conversion to the instrument-specific format begins. This process consists of the translation of the Turtle file into a set of Pure Data files. Pure Data files can be easily created using the textual syntax of the program, which allows to specify objects, sub-patches, and their connections. Notably, for simplicity the translation process assumes that the smart guitar already possesses an implementation of the digital audio effect (the delay effect, as indicated also in listing 4). In the contrary case, it is possible to implement a mechanism for downloading from a server the missing files related to a digital audio effect. Fig. IV shows the generated Pure Data code and the corresponding graphical representation of the patch.

Listing 1. Portion of the JSON configuration file for the Elk Audio OS sound engine.

```
"tracks" : [
    {
        "name" : "main",
        "mode" : "stereo",
        "inputs" : [
            {
                "engine_bus" : 0,
                "track_bus" : 0
            }
        ],
        "outputs" : [
            {
                "engine_bus" : 0,
                "track_bus" : 0
            }
        ],
        "plugins" : [
            {
                "path" : "/udata/plugins/mdaDelay.so",
                "name" : "mdaDelay",
                "type" : "vst2x"
            }
        ]
    }
]
```

[2]https://github.com/lucaturchet/OBDA_based_SMIs_preset_exchange/



```
#N canvas 622 23 550 412 12;
#X obj 7 13 adc~ 1 2;
#X obj 7 369 dac~ 1 2;
#X obj 158 13 adc~ 3;
#X obj 7 337 _delay_feedback, f 22;
#X obj 158 111 lop~;
#X obj 158 147 snapshot~;
#X obj 158 274 change;
#X obj 158 232 maxlib/scale 0.3 0.6 0 1 0;
#X obj 158 193 clip 0.3 0.6;
#X msg 183 80 50;
#X obj 385 12 loadbang;
#X msg 385 38 1;
#X obj 385 71 metro 100;
#X connect 0 0 3 0;
#X connect 0 1 3 0;
#X connect 2 0 4 0;
#X connect 3 0 1 0;
#X connect 3 0 1 1;
#X connect 4 0 5 0;
#X connect 5 0 8 0;
#X connect 6 0 3 2;
#X connect 7 0 6 0;
#X connect 8 0 7 0;
#X connect 9 0 4 1;
#X connect 10 0 11 0;
#X connect 10 0 9 0;
#X connect 11 0 12 0;
#X connect 12 0 5 0;
```

Fig. 3. The generated Pure Data code and its graphical representation as a patch (comments added afterwards for clarity's sake).

Listing 2. Portion of the JSON configuration file for the Elk Audio OS sensor engine.

```
"sensors" : [
  {
    "id" : 1,
    "enabled": true,
    "name" : "pressure_sensor_0",
    "sensor_type" : "analog_input",
    "mode" : "on_value_changed",
    "hardware" :
    {
      "hardware_type" : "analog_input_pin",
      "pins" : [0],
      "delta_ticks" : 1,
      "adc_resolution" : 8,
      "filter_time_constant" : 0.020
    }
  }
]
```

Listing 3. Portion of the JSON configuration file for the sensor to sound-parameter mappings.

```
{
    "mappings": [
        {
        "mapping_id" : 1,
        "sensor_id" : 1,
         "name_effect" : "mdaDelay",
         "effect_parameter" : "Feedback"
        }
    ]
}
```

V. Discussion and Conclusion

In this paper we presented a system for SMIs preset sharing based on the ontology-based data access paradigm. The proposed approach allows one to share presets between heterogeneous SMIs by mapping information about the configuration of an instrument to the concepts of the SMI Ontology [8]. Thanks to this approach, SMIs developers can implement programs that convert proprietary formats for the configuration of the instrument into a common format for SMIs, and vice versa. As a proof of concept we presented a concrete implementation of the proposed framework, where two heterogeneous smart guitars could exchange a file format for their configuration. The workflow could be completely automated and the involved software could efficiently run on embedded systems with relatively low computational capacity.

Recently, the MIDI 2.0 protocol has been developed and licensed. MIDI 2.0 represents an industry effort to introduce a semantic layer in the communication among musical devices. Specifically, MIDI 2.0 is based on the features of Profile Configuration and Property Exchange (the latter describable in a JSON file), which may be potentially useful to configure an SMI. Nevertheless, it is important to note that the MIDI 2.0 and its ecosystem differ from those of SMIs. In this study we aimed to develop an SMI-specific solution to the issue of preset sharing. Our approach mainly differentiates from MIDI 2.0 for the fact that we leverage a shared ontology and that it relies on Semantic Web technologies, whereas MIDI 2.0 is not ontology-based and does not exploit the power of linked data. We believe that the adoption of a shared representation like the SMI ontology is a powerful tool to facilitate interoperability between heterogeneous SMIs, and in particular to solve the issue of preset sharing.

In principle, the MIDI 2.0 Property Exchange definitions could also leverage concepts and relationships described in the SMI Ontology, and potentially a MIDI 2.0-based preset sharing mechanism for SMIs could be envisioned. However, the architectures of Bela and Elk are not yet fully equipped with MIDI 2.0 support mechanisms that enable the specific configurations of the respective sound engines and sensor interfaces. More importantly, MIDI 2.0 development frameworks are just in their infancy and there are not commonly available software tools that facilitate their wide adoption. All these aspects motivated us to develop a specific preset sharing mechanism for SMIs for such platforms.

Notably, it is important to acknowledge that this study has the following limitation. The SMI Ontology is an implementation-driven ontology that evolves during its use while developing SMIs and SMIs-based applications [8]. As a consequence, there is a risk that the current definition of the mappings between local and global schemas required by an OBDA approach could change in the future. This may introduce complexities that it is necessary to handle in a satisfactory way in order to avoid inconsistencies as the ontologies changes and grow. As noted by Lembo et al. it is important not to underestimate such a risk [22].

As new SMIs appear on the market, novel proprietary formats may emerge from the musical instruments manufacturing industry. Artists may be willing to share the configuration of their instruments with those fans interested in playing their songs, understanding its structure, or reusing it for their purposes. Therefore, it is plausible to expect a future need of a common conversion system across such formats. This paper was conceived to provide a concrete method to cater to such need. Notably, while the proposed approach was specifically devised for SMIs it is in principle possible to apply its general structure to other musical interfaces in need of exchanging content (such as configurations of virtual reality headsets or haptic devices involved in musical activities). This calls for further research in the field of the IoMusT.

References

[1] L. Turchet, "Smart Musical Instruments: vision, design principles, and future directions," *IEEE Access*, vol. 7, pp. 8944–8963, 2019.

Listing 4. Portion of the Turtle file resulting from the mapping process.

```
@prefix ex:       <http://www.example.audio/ex> .
@prefix smi:      <http://purl.org/ontology/iomust/smi#> .
@prefix iomust:   <http://purl.org/ontology/iomust/internet_of_things/iomust> .
@prefix mo:       <http://purl.org/ontology/mo/> .
@prefix sosa:     <http://www.w3.org/ns/sosa/> .
@prefix mx:       <http://purl.org/ontology/studio/mixer/> .
@prefix con:      <http://purl.org/ontology/studio/connectivity#> .
@prefix device:   <http://purl.org/ontology/studio/device/> .
@prefix fx:       <https://w3id.org/aufx/ontology/1.0> .
@prefix studio:   <http://purl.org/ontology/studio/> .
@prefix foaf:     <http://xmlns.com/foaf/0.1/> .
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

ex:ElkGuitar                rdf:type                  iomust:SmartInstrument ,
                                                      smi:SmartGuitar ;
                            smi:smi_component         ex:ElkGuitarSoundEngine .
ex:ElkGuitarSoundEngine     rdf:type                  smi:SoundEngine ;
                            device:component          ex:ElkGuitarSoftwareMixer .
ex:ElkGuitarSoftwareMixer   rdf:type                  mx:SoftwareMixer ;
                            mx:channel                ex:ElkGuitarCh1 .
ex:ElkGuitarCh1             rdf:type                  mx:SteroChannel .
                            fx:name                   "main" ;
                            mx:input                  con:AnalogueInput;
                            mx:output                 con:AnalogueOutput;
                            smi:hosts_plugin          ex:Delay1.
ex:Delay1                   rdf:type                  fx:PlugIn,
                                                      studio:Delay;
                            smi:plugin_name           "mdaDelay";
                            fx:PlugInAPI              "vst2x";
                            fx:has_parameter          ex:Delay1FeedbackPar .
ex:Delay1FeedbackPar        rdf:type                  fx:NumericParameter;
                            fx:name                   "Feedback" .
ex:PressureSensor1          rdf:type                  smi:PressureSensor ;
                            smi:mapping_function      ex:mapping1 .
ex:mapping1                 rdf:type                  smi:MappingTransform ;
                            smi:maps                  ex:Delay1FeedbackPar ;
                            smi:mapping_function_type "linear" .
```

[2] L. Turchet, A. McPherson, and M. Barthet, "Co-design of a Smart Cajón," *Journal of the Audio Engineering Society*, vol. 66, no. 4, pp. 220–230, 2018.

[3] L. Turchet, M. Benincaso, and C. Fischione, "Examples of use cases with smart instruments," in *Proceedings of Audio Mostly Conference*, 2017, pp. 47:1–47:5.

[4] L. Turchet, C. Fischione, G. Essl, D. Keller, and M. Barthet, "Internet of Musical Things: Vision and Challenges," *IEEE Access*, vol. 6, pp. 61 994–62 017, 2018.

[5] L. Turchet and P. Kudumakis, "Requirements for a file format for smart musical instruments," in *Proceedings of the Workshop on Multilayer Music Representation and Processing*, 2019, pp. 5–9.

[6] A. Baratè, G. Haus, and L. Ludovico, "A critical review of the IEEE 1599 standard," *Computer Standards & Interfaces*, vol. 46, pp. 46–51, 2016.

[7] I. Jang, P. Kudumakis, M. Sandler, and K. Kang, "The MPEG Interactive Music Application Format Standard [Standards in a Nutshell]," *IEEE Signal Processing Magazine*, vol. 28, no. 1, pp. 150–154, 2011.

[8] L. Turchet, P. Bouquet, A. Molinari, and G. Fazekas, "The smart musical instruments ontology," *Journal of Web Semantics*, 2021.

[9] A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, "Linking data to ontologies," *Journal on data semantics X*, pp. 133–173, 2008.

[10] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, and M. Zakharyaschev, "Ontology-based data access: A survey," in *In Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI/AAAI, 2018, pp. 5511–5519.

[11] D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, M. Rodriguez-Muro, R. Rosati, M. Ruzzi, and D. F. Savo, "The mastro system for ontology-based data access," *Semantic Web*, vol. 2, no. 1, pp. 43–53, 2011.

[12] M. Rodriguez-Muro, R. Kontchakov, and M. Zakharyaschev, "Ontology-based data access: Ontop of databases," in *International Semantic Web Conference*. Springer, 2013, pp. 558–573.

[13] A. McPherson and V. Zappi, "An environment for Submillisecond-Latency audio and sensor processing on BeagleBone black," in *Audio Engineering Society Convention 138*. Audio Engineering Society, 2015.

[14] L. Turchet and C. Fischione, "Elk Audio OS: an open source operating system for the Internet of Musical Things," *ACM Transactions on the Internet of Things*, vol. 2, no. 2, pp. 1–18, 2021.

[15] A. Hunt, M. Wanderley, and M. Paradis, "The Importance of Parameter Mapping in Electronic Instrument Design," in *Proceedings of the Conference on New Interfaces for Musical Expression*, 2002.

[16] K. Janowicz, A. Haller, S. J. Cox, D. Le Phuoc, and M. Lefrançois, "SOSA: A lightweight ontology for sensors, observations, samples, and actuators," *Journal of Web Semantics*, 2018.

[17] T. Wilmering, G. Fazekas, and M. Sandler, *AUFX-O: Novel Methods for the Representation of Audio Processing Workflows*, ser. Lecture Notes in Computer Science,. Springer, Cham, 2016, vol. 9982.

[18] Y. Raimond, S. Abdallah, M. Sandler, and F. Giasson, "The music ontology," in *Proceedings of International Society for Music Information Retrieval Conference*, 2007.

[19] G. Fazekas and M. Sandler, "The Studio Ontology Framework," in *Proceedings of the International Society for Music Information Retrieval conference*, 2011, pp. 24–28.

[20] L. Turchet, F. Antoniazzi, F. Viola, F. Giunchiglia, and G. Fazekas, "The internet of musical things ontology," *Journal of Web Semantics*, vol. 60, p. 100548, 2020.

[21] L. Turchet, G. Zhu, and P. Bouquet, "Populating the smart musical instruments ontology with data," in *2020 27th Conference of Open Innovations Association (FRUCT)*. IEEE, 2020, pp. 260–267.

[22] D. Lembo, J. Mora, R. Rosati, D. F. Savo, and E. Thorstensen, "Towards mapping analysis in ontology-based data access," in *International Conference on Web Reasoning and Rule Systems*. Springer, 2014, pp. 108–123.