Semantic Web of Musical Things: achieving interoperability in the Internet of Musical Things

Luca Turchet[a,*], Francesco Antoniazzi[b]

[a]*Department of Information Engineering and Computer Science, University of Trento*
[b]*Independent Researcher*

Abstract

The Internet of Musical Things (IoMusT) refers to the extension of the Internet of Things paradigm to the musical domain. Interoperability represents a central issue within this domain, where heterogeneous Musical Things serving radically different purposes are envisioned to communicate between each other. Automatic discovery of resources is also a desirable feature in IoMusT ecosystems. However, the existing musical protocols are not adequate to support discoverability and interoperability across the wide heterogeneity of Musical Things, as they are typically not flexible, lack high resolution, are not equipped with inference mechanisms that could exploit on board the information on the whole application environment. Besides, they hardly ever support easy integration with the Web. In addition, IoMusT applications are often characterized by strict requirements in terms of latency of the exchanged messages. Semantic Web of Things technologies have the potential to overcome the limitations of existing musical protocols by enabling discoverability and interoperability across heterogeneous Musical Things. In this paper we propose the Musical Semantic Event Processing Architecture (MUSEPA), a semantically-based architecture designed to meet the IoMusT requirements of low-latency communication, discoverability, interoperability, and automatic inference. The architecture is based on the CoAP protocol, a semantic publish/subscribe broker, and the adoption of shared ontologies for describing Musical Things and their interactions. The code implementing MUSEPA can be accessed at: `https://github.com/CIMIL/MUSEPA/`.

Keywords: Internet of Musical Things, Web of Things, Smart Musical Instruments, Ontologies, Semantic Audio
2010 MSC: 00-01, 99-00

1. Introduction

The Internet of Musical Things (IoMusT) refers to the extension of the Internet of Things (IoT) paradigm to the musical domain [1]. This is an emerging and rapidly evolving field, as demonstrated by a growing academic literature corpus (see e.g., [2, 3, 4, 5, 6, 7]) and the appearance of a number of products developed by the music technology industry (see e.g., the Elk Audio OS developed by Elk[1] [8] or the HyVibe Smart Guitar by HyVibe[2]). In the IoMusT, objects dedicated to the production and/or reception of musical content (Musical Things) are connected by a networked infrastructure that enables multidirectional communication, both locally and remotely, between different stakeholders such as audience members, composers, performers, live sound engineers, studio producers, as well as music teachers and music students. The ecosystems that will form around Internet of Musical Things technologies are envisioned to support novel forms of interactions between such stakeholders by means of novel musical applications and services [9, 10]. This has the potential to revolutionize the way music is composed, performed, experienced, learned and recorded.

Interoperability represents a central issue within this domain, where heterogeneous Musical Things are envisioned to communicate between each other. Musical Things, such as smart musical instruments [11] or musical haptic wearables [12, 13], may serve radically different purposes within an IoMusT ecosystem (e.g., from generation of haptic stimuli to real-time analysis of musical content) and to accomplish the envisioned novel interactions between the various stakeholders they need to communicate through a common language. Automatic discoverability of resources in the network [14] is also a desirable feature in IoMusT ecosystems. In addition, IoMusT applications are often characterized by strict requirements in terms of latency of the exchanged messages. To date, interoperability across musical devices has mostly relied on protocols for the exchange of musical messages such as Musical Instrument Digital Interface (MIDI) or Open Sound Control (OSC) [15] and tools based on it (e.g., Libmapper [16]). However, the existing musical protocols are not adequate to support discoverability and interoperability across the wide heterogeneity of Musical Things, as they are typically not flexible, lack high resolution, are not equipped with inference mechanisms, and do not support easy integration

*Corresponding author
Email address: `luca.turchet@unitn.it` (Luca Turchet)
[1]https://elk.audio/
[2]https://www.hyvibe.audio/smart-guitar/

with the Web [17].

Semantic technologies, such as Semantic Web [18], knowledge representation [19], and semantically-based communication architectures [20], have been envisioned as a solution to enable interoperability across heterogeneous Musical Things as they have the potential to overcome the limitations of existing protocols conceived for musical contexts [1, 17]. The Semantic Web was born to transform the Web from a repository of human-readable information into an entity that allows for the machine-understandability of data. This vision has spurred the emergence of novel computing paradigms such as the Semantic Web of Things (SWoT) [21, 22]. SWoT has recently appeared as the latest evolution of the IoT and, as the name suggests, requires that devices interoperate through the Internet using Web protocols and standards coming from the Semantic Web. Whereas the Web of Things (WoT) [3] refers to a computing paradigm where everyday objects are fully integrated with the Web [23], the SWoT relies on a WoT implementation crafted on the technologies proposed in the so-called "Semantic Web stack". The main components of this stack allow for: i) the univocal identification of resources thanks to IRI (International Resource Identifiers); ii) the representation of data as a set of triples thanks to RDF (Resource Description Framework) [24]; iii) the definition of an ontology to clearly state the meaning of represented data by using RDFS (RDF Schema) [25] and OWL (Web Ontology Language) [26]; iv) the storage and retrieval of data via the SPARQL Update [27] and Query languages [28].

In the last decade, the specification and implementation of novel Application Programming Interfaces (APIs) in web browsers (such as WebAudio, WebSockets or WebGL) has enabled the use of the web platform as a fertile playground for musicians [29]. The inherent networked nature and scalability of web technologies has allowed one to simplify and democratize the use of mobile devices for musical purposes [30], and in particular in performance contexts [31]. On the other hand, various ontologies specific to the musical domain have been built in recent years (see e.g., the Music Ontology [32] or the Studio Ontology [33]). Nevertheless, thus far little attention has been devoted by researchers to the application of Semantic Web technologies to live music contexts and even less to the IoMusT scenarios. Differently from other Things within the IoT, to date the few existing Musical Things are disconnected from the Web and form a myriad of small incompatible islands. The possibility to monitor and control Musical Things via the Web, especially in real-time, using a semantically enriched common language and dedicated semantic architectures unleashes the potential to explore novel artistic avenues, such as performance and composition [3], for instance based on emergent properties of an IoMusT ecosystem [34, 35, 36]. Indeed, semantic technologies based on ontologies as well as communication architectures specific to the IoMusT domain can assist in managing, querying, and combining information characterizing an IoMusT ecosystem, including data about the music produced, the involved stakeholders, the utilized Musical Things and their application and services [17].

In this paper, we present a working implementation of the SWoT applied to the musical domain through the adoption of shared ontologies for describing Musical Things and their interactions, coining this endeavor as the "Semantic Web of Musical Things". Specifically, we present a novel semantic communication architecture based on the Constrained Application Protocol, which exploits the combination of existing ontologies that include patterns for dynamic interactions between Musical Things [37, 17, 38]. Especially in live music scenarios, IoMusT deployments are often characterized by dynamicity, such as the need of adding or removing new Musical Things joining or leaving the ecosystem, re-defining the Musical Things' behavior, or tuning the ecosystem parameters. A relevant research challenge is how to support the IoMusT deployment reconfiguration seamlessly, i.e., avoiding the manual intervention during a live music performance. Moreover, some IoMusT deployments have strict requirements in terms of latency of the exchanged messages and support for constrained devices. A second challenge is thus how to ensure that communication among Musical Things occurs within acceptable latencies.

In developing our architecture we focused on the following research questions:

1. How to ease the discovery and the management of Musical Things?;
2. How to support the dynamic reconfiguration of IoMusT ecosystems, e.g., the deployment of new Musical Things or the interconnection among the existing ones, while minimizing the need of manual configuration (for system administrators) and coding (for programmers)?;
3. How to ensure low latency communication of semantically-based messages?

It is worth noticing that our architecture is not meant to support continuous real-time interactions across local or remote devices (e.g., the continuous control of a parameter of a synthesizer via the knob of a wirelessly connected controller). It is rather conceived to address the three research questions mentioned earlier, which are based on discrete interactions (e.g., the fast discovery of when a new Musical Thing joins the ecosystem or the fast automatic reconfiguration of the ecosystem based on inference mechanisms).

To assess the developed architecture we implemented an IoMusT ecosystem based on the Semantic Web of Musical Things, showing how the dynamic setup proposed can foster interoperability at information level allowing smart discovery as well as enabling orchestration and automatic interaction through the available semantic information, while at the same time guaranteeing acceptable latencies for musical communication. Ultimately, we demonstrate

[3]https://www.w3.org/WoT/

that these features allow one to create a novel class of IoMusT ecosystems supporting different types of interactions between various stakeholders.

2. Related works

2.1. Semantic Web of Things architectures

Relatively few recent works in literature propose IoT architectures mediated by semantics, for instance in context of smart agriculture [39, 40], smart cities [41, 42] and healthcare [43, 44, 45]. One noticeable example is the study reported in [37], where a global idea of *Semantic Web of Things* mediated by a publish-subscribe architecture is developed, to be intended as one of the origins of this work. The authors reported a working implementation of the WoT declined in its semantic flavor through the adoption of a shared ontology for describing devices. Such an ontology, named SWOT, accomplishes a high-level abstraction of the devices and of their capabilities leveraging the concept of Thing Description proposed in [46] for the WoT and the concepts Property, Action, and Event proposed in [47]. Furthermore, reporting the contents of the aforementioned [37], the study also addressed one of the main limitations present in the SWoT, the fact that ontologies and semantic-formatted data are considered to be static, while any real context is continuously evolving dynamically. This limitation was overcome by equipping SWOT not only with the tools to build a static description of the things, but also with a set of concepts that regulate the dynamic interaction. In this way, the dynamic nature of the ontology allows one not only to describe the abstract context, but also to follow its real-time evolution. SWOT was used in conjunction with a SPARQL Event Processing Architecture (SEPA) [48, 20]. SEPA aims to enhance triple stores with a publish-subscribe layer on top the SPARQL 1.1 protocol. SEPA clients, then, by using SPARQL 1.1 subscribe and update languages can, respectively, subscribe to and publish semantic data. In this way, SEPA allows one to easily create a semantic representation of the context and keep it coherent with the physical environment as time passes.

Other works in the panorama suggest approaches that are different from the one here presented. Semantic Stream Processing, for instance, in [49, 50, 51, 52, 53] and [54] (just to name a few), where Bermudez-Edo et al. describe an instantiation of SSN ontology [55] extended to stream annotation. They refer to streams of RDF data to elaborate events, which is not our case since we let our environment evolve handling events and situations through a publish-subscribe architecture.

Additionally, [56] presents an attempt to realize a triple store for limited memory devices based on the same CoAP protocol we use in this work. However, as already mentioned, our target is not to realize a conventional triple store, but an architecture for musical applications enriched with semantics.

Eventually, it is possible to cite all the works, drafts and recommendations made by W3C and their derivations on IoT and WoT, some examples of which are [57, 58, 59, 60]. The main difference with the present work, indeed, is that the main topic here are musical applications on top of the aforementioned semantic publish-subscribe.

2.2. Musical Things

In the IoMusT vision [1] a Musical Thing can take the form of a musical instrument, a wearable, a smartphone, or any other smart device utilized to control, generate, or track responses to musical content. At the heart of a Musical Thing there is an embedded system dedicated to audio and/or sensor processing tasks, and various platforms have been proposed for this purpose [61]. In the past few years various studies have investigated the use of embedded systems serving musical applications in networked settings (e.g., using WebAudio technologies [3]).

One of the building blocks of the IoMusT is represented by the emerging family of musical instruments termed "smart musical instruments" [11]. Such instruments are the result of the integration of embedded computational intelligence (running on dedicated platforms, e.g., [62]), wireless connectivity, embedded sound delivery system, and an onboard system for feedback to the player. They offer direct point-to-point communication between each other and other portable sensor-enabled devices connected to local networks and to the Internet, fostering networked music performance systems [63, 64]. To date, only a handful of instruments with such characteristics exist. Examples include the Sensus Smart Guitar developed by Elk[4], the Smart Mandolin described in [65], or the Smart Cajón reported in [66]. A growing literature of applications and services for them is emerging as the result of the novel capabilities they afford, such as the interconnection of a smart guitar with mobile phones for collaborative music making over a local wireless network [67] or the use of distributed intelligence, via cloud computing and edge computing paradigms, for music learning and improvisation contexts [7].

Another type of Musical Things is the class of devices termed "musical haptic wearables". These are wearable devices for performers [12] and audience members [13], which encompass haptic stimulation, gesture tracking, and wireless connectivity features. Musical haptic wearables were devised to enrich musical experiences by leveraging the sense of touch as well as providing new capabilities for creative participation. Their conception was grounded on the findings of research in the field of haptic technologies developed for musical applications [68] and of participatory live music performances [69].

2.3. Interoperability in the musical domain

The MIDI protocol is a well established framework enabling Digital Musical Instruments [70] to exchange mu-

[4]https://elk.audio/sensus-smart-guitar/

sical information. It was conceived in the 80s to enable interoperability across musical instruments developed by different vendors. MIDI is not well suited to achieve interoperability across heterogeneous devices such those of the IoMusT because it was specifically conceived for communication across musical instruments. In addition, MIDI is very limited in resolution (e.g., it uses integers between 0 and 127), which prevents to represent information with a high level of detail and accuracy.

In a different vein, OSC is a protocol more flexible and with higher resolution than MIDI, as it enables user-defined namespaces and supports messages with various formats (including floats and strings). This would make OSC more suitable to facilitate communication across heterogeneous Musical Things. However, OSC is not equipped with standard namespaces for interfacing devices and as a consequence, connected devices neither know each other or each other's capabilities [71]. In the past decade, various tools based on OSC have been proposed where a semantic layer is added to the conventional OSC protocol structure (such as Libmapper and Sense-World DataNetwork [72, 73, 74]). These protocols provide decentralized resource allocation and discovery, and flexible connectivity letting devices describe themselves and their own capabilities. Nevertheless, they target the use of a LAN subnet where support for multicast can be guaranteed [16]. They are not conceived for Web-based interactions nor they support mechanisms of automatic inference.

Semantic Web standards allow one to rely on a common representation of data based on shared and agreed ontologies, and provide a powerful mechanism to solve the problems of discoverability and orchestration of services. However, to date few attempts have been made to apply Semantic Web technologies to live music scenarios and IoMusT ecosystems to achieve interoperability across devices. The first effort towards this direction is represented by the semantically-enriched IoMusT architecture reported in [75], which relies on a semantic audio server and edge computing techniques. Specifically, the SEPA architecture [20] was utilized as an interoperability enabler allowing multiple prototypes of Musical Things to cooperate, relying on a music-related ontology. A limitation of the developed architecture was the involvement of an ontology restricted to the representation of simple musical features, which prevented Musical Things dedicated to purposes other than music generation to join the ecosystem formed around the architecture. For this purpose, two ontologies have been recently devised: the Internet of Musical Things Ontology [17] and the Smart Musical Instruments Ontology [38] that make large use of standard lower-level ontologies like SOSA/SSN [55], PROV [76], and many others to create their own concepts. Nevertheless, they have not been employed yet in an actual IoMusT ecosystem, an endeavor tackled in the present study.

However, Semantic Web technologies are in general slow and verbose, so not suitable for IoT applications relying on real-time aspects such as those envisioned in the IoMusT. Indeed the Semantic Web stack is oriented towards more static scenarios, where information evolves at a lower rate [20]. To cope with this issue, the authors created C Minor [77]. Such an architecture improved the system reported in [75] by using a lightweight IoT protocol for machine-to-machine communication, the Constrained Application Protocol (CoAP) [78]. The proposed approach combines the expressive power of Semantic Web technologies (i.e., RDF, RDFS, OWL and SPARQL) with the advantages of a top-class IoT protocol such as CoAP (binary data sent over UDP, resource/observe interaction pattern). The technology proposed was envisioned to be useful in IoMusT scenarios where several actors are involved in the music creation process, such as audience members using large-scale participatory live music systems. In such scenarios it is indeed essential to reduce the transmission bandwidth consumption. However, C Minor was neither conceived for nor applied within an actual IoMusT ecosystem characterized by dynamic interactions, nor leveraged an ontology specific to the IoMusT. These aspects are the object of the present study.

3. Musical Semantic Event Processing Architecture

To implement our vision of the Semantic Web of Musical Things, we propose the "Musical Semantic Event Processing Architecture" (MUSEPA). This architecture leverages several concepts underlying the SEPA [48, 20] and C Minor architectures [77], and applies such concepts to complex IoMusT ecosystems. Specifically, MUSEPA is a semantic client/server architecture where clients communicate by means of messages exchanged through a server (also named broker). The main assignment of the broker is to hold and provide access to an RDF knowledge base by means of the SPARQL query and update language. The broker is then a SPARQL endpoint and as such, clients communicate through SPARQL requests. Nevertheless, differently from a standard SPARQL endpoint, MUSEPA also implements the publish/subscribe paradigm: subscriptions allow clients to be notified about changes in the knowledge base avoiding polling. MUSEPA then provides the ability to retrieve data from a knowledge base through the request/response or publish/subscribe paradigm and update the knowledge base by adding, modifying or deleting triples from it. Fig. 1 illustrates examples of data flow among clients and broker.

Notably, by using a publish/subscribe semantic endpoint like MUSEPA, we ensure that the knowledge base is constantly updated with the context evolution. Through MUSEPA, clients avoid polling for data (i.e., data is dispatched as soon as available). Within SEPA and MUSEPA a subscription is identified by a SPARQL query, that represents a subgraph pattern by means of the variables composing the WHERE clause. On top of this, both architectures would monitor the changes happening in the RDF store over the subgraphs matching the pattern, and notify with information on the ones that were modified. This is precisely where the main difference between MUSEPA and SEPA lies: MUSEPA does not calculate the added and removed triples like SEPA does, attempting in this way to address SEPA's heaviest performance bottleneck. Please refer to Section 3.2 for a better understanding of the differences between SEPA's and MUSEPA's internal engines.

Leveraging the SWoT consists in setting up ontologies to regulate the relations among the Web resources, and therefore their differentiation based on their reciprocal connections. For this purpose, MUSEPA utilizes simultaneously the SWOT ontology [37], the IoMusT ontology [17], and the SMI ontology [38]. In this research we took as an assumption that devices would share the same ontological pattern description, which is actually a necessary compatibility requirement. The utilization of a publish/subscribe semantic broker in conjunction with both an ontology supporting the representation of the dynamic behavior of devices and ontologies describing Musical Things, allows one to achieve semantic representation of the context of an IoMusT ecosystem and keep it coherent with its evolution.

As a matter of fact, the concurrent usage of MUSEPA and the SWOT ontology, as explained in detail in [37], enriches the environment with plain IoT semantic knowledge and its Action-Event-Property architecture. Moreover, it realizes over the publish-subscribe mechanism an environment capable of reacting to and interacting with the events that occur, and therefore addresses the dynamic reconfiguration of the environment. Such reconfiguration in our case is mediated by the specific IoT application semantics, that is the musical IoT provided through the IoMusT and SMI ontologies. These ontologies allow the process of semantic discovery, which we can define as the activity of becoming aware of which items the IoT environment contains, with specific features that could help a Musical Thing to achieve its goal.

MUSEPA is structured around the CoAP protocol for the exchange of messages, which presents several benefits compared to available alternatives such as MQTT and AMQP:

- It was proposed by the Internet Engineering Task Force, and in particular the Constrained RESTful Environments subgroup, as a standard Request For Comment. Therefore, it is an open, fully documented specification [79], which can facilitate interoperability in the IoT;

- CoAP headers have a minimum impact on the message size, [80];

- CoAP is based on UDP (while MQTT and AMQP rely on TCP), but still supports retransmission of lost or damaged packets through the mechanism of confirmable and non confirmable messages. In this way CoAP removes the overhead caused by the three-way handshake protocol;

- Compared to MQTT and AMQP, CoAP presents the lowest bandwidth requirement and the lowest latency [80];

- CoAP addresses the problem of discoverability [81] by providing a list of the available resources;

- CoAP is designed to be easily mapped on HTTP, then binding the SPARQL 1.1 Protocol [82] to CoAP is a very straight-forward process.

In developing MUSEPA we considered two kinds of IoMusT ecosystems. The first kind concerns ecosystems in which low-latency communication is a crucial aspect (e.g., $<$ 100 milliseconds, depending on the application at hand). To this category belong ecosystems based on applications for real-time networked music performances [63, 64] and in general real-time interactive musical systems. The second kind regards those ecosystems in which the communication may be asynchronous or real-time but tolerating relatively large latencies. This is the case of ecosystems dedicated to musical activities such as pedagogy or composition, where services do not strictly require low-latency communications. Accordingly, a relevant design choice consisted in the capability of the system of storing and restoring a triplestore.

In the context in which low-latency communication is crucial we did not rely on the recovery of triples from a triplestore (e.g., Apache Jena Fuseki), given the fact that its use would cause a slower interaction due to its internal functioning (see Section 4.3). Instead, we utilized a simple system in which an RDF graph is populated and temporarily saved in local memory (specifically, for this purpose we used the Python library RDFLib), which is much faster in handling requests and subscriptions. Moreover, in taking this design choice we considered that the utility of restoring triples from a triplestore during live music scenarios may be less relevant, as well as keeping them in a persistent storage. Conversely, a triplestore was utilized for those contexts in which the latency is not a strict requirement, and when the recovery of the triples created in past utilizations of the system is an important aspect.

3.1. From HTTP to CoAP

The SPARQL Event Processing Architecture (SEPA), that is the ancestor of C Minor formerly, and indeed of MUSEPA, was developed on top of HTTP and WebSocket protocols. SEPA expects an HTTP call to perform queries and updates, and keeps a number of open WebSockets equal to the number of active subscriptions. This choice was already addressed in the research that ended up with the presentation of C Minor [77]: both WebSocket and HTTP protocols exploit TCP. While it provides a reliable handshake, TCP may not be the best choice in the case of constrained devices, or specific application. A typical example of this is the streaming of videos made on a UDP transport layer, since the rate of data transmitted is more

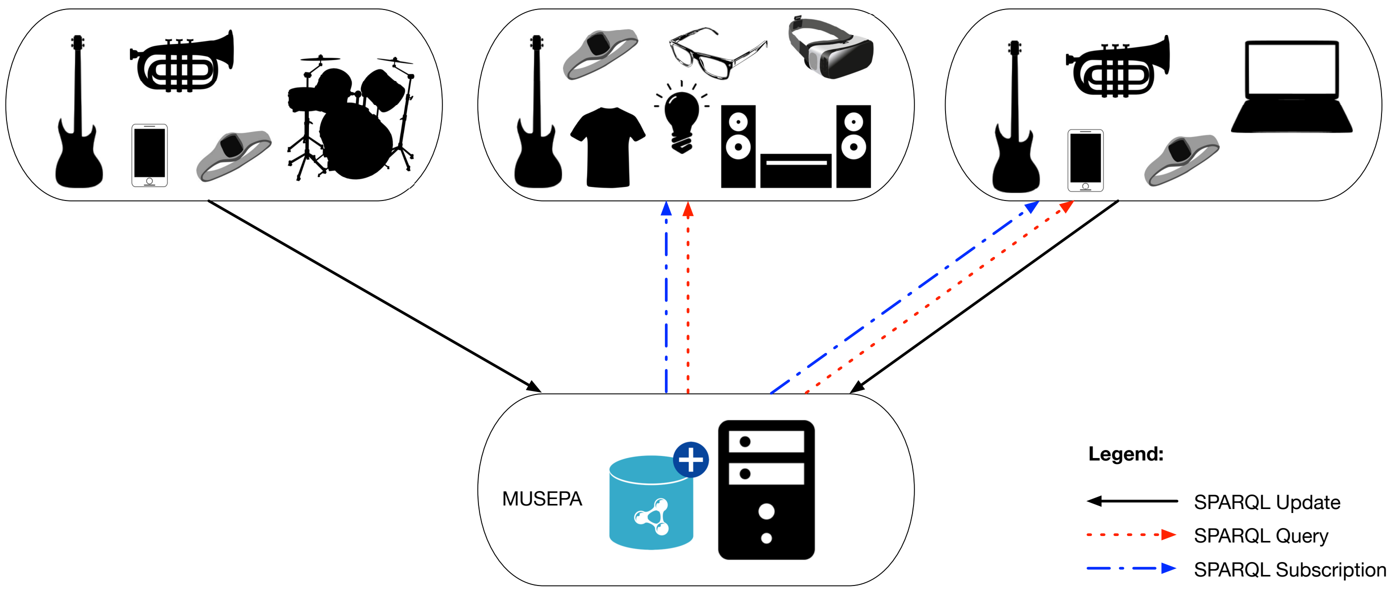

Figure 1: Schematic representation of an IoMusT ecosystem based on the proposed semantic architecture. Different types of Musical Things can perform different actions: a client can only update the knowledge base (left), a client can explore the knowledge base (middle), a client performs both update and exploration (right).

important than to ensure that all the packets will be successfully delivered.

MUSEPA (and previously C Minor) uses the CoAP protocol to build an architecture similar to SEPA. While the effective differences in inner mechanisms will be discussed in the next paragraphs, it is worth noticing that, first of all, we are not using here two different protocols (HTTP and WebSocket), but just one (CoAP) to obtain both the query/update interaction and the subscription. This is a relevant point, if we consider a typical problem that IoT programmers must solve: the limited memory in devices. This issue often appears as an error while uploading the firmware to the device, due to the fact that too much place is required to include the needed libraries. By using one unique protocol, we ensure that only one library is required to run a minimal MUSEPA client.

Table 1 explains how MUSEPA can be used by the clients. Namely, it states that aside from the Update and Query requests, that are quite easy to understand, there are two additional requests: *subscription* and *observation*. The latter, in particular, is closely related to the CoAP protocol `OBSERVE` feature that hereby allows us to re-create a function similar to the WebSocket subscription in SEPA.

3.2. Software architecture

The code that implements MUSEPA can be found in the Github repository[5], together with a full explanation on how to download or clone it, and then run it on a Linux machine. MUSEPA, like the ontologies that have been named in the text, are all free and publicly available.

[5]`https://github.com/CIMIL/MUSEPA/`

MUSEPA, as it appears from the code and from Fig. 1, has been implemented in Python 3 using the `aiocoap` library. The architecture is overall composed of a central CoAP server which is the MUSEPA engine: it will deal with direct client interaction, and provide a publish-subscribe environment to them. On a lower level, an RDF triple store is required, i.e., the location in which all semantic information about the applications will be stored. To interact and store the knowledge base in an RDF-compliant format, three possibilities have been realized:

1. RDFLib v5.0.0 Python3 library was used to create a volatile (but more performant) RDF knowledge base;
2. The interaction with Blazegraph has been implemented and is also possible (external endpoint), to have a persistent knowledge base;
3. Interaction with Apache Jena Fuseki is also possible (external endpoint), similarly, and provides a persistent knowledge base.

The choice of the endpoint is irrelevant (excepted from the performance point of view), once a client starts an interaction with MUSEPA.

As it can be seen in Fig. 2, various resources are available in the MUSEPA server. Some of them come directly from the previous C Minor implementation, although a few differences will be listed in the next paragraphs.

First of all, let us consider a simple case in which MUSEPA runs locally. In this situation the `Root` resource could be renamed as `localhost`. Therefore, any client would interact with MUSEPA as explained in Table 1:

- a `CoAP GET` to `coap://localhost/sparql/query`, providing a SPARQL Query payload, to get a query result;

MUSEPA CoAP requests		**Notes**
Update Request verb	`POST`	
Text specified as:	payload	SPARQL 1.1 Update or Turtle
Status code for success:	`2.04`	`CHANGED`
Status code for error:	`4.00` or `4.02`	`BAD REQUEST`, `BAD OPTION`
Response:	No payload	
Query Request verb	`GET`	
Text specified as:	payload	SPARQL Query
Status code for success:	`2.05`	`CONTENT`
Status code for error:	`4.00` or `4.02`	`BAD REQUEST`, `BAD OPTION`
Response:	payload	JSON query answer
Subscription Request verb	`POST`	`GET` (for information)
Text specified as:	payload	SPARQL Query
Status code for success:	`2.01`	`CREATED`
Status code for error:	`4.02`	`BAD OPTION`
Response:	payload	Observable resource name
Observation Request verb	`GET`	`OBSERVE/UNOBSERVE`
Text specified as:	No payload	
Status code for success:	`2.04`, `2.05`, `2.02`	`CHANGED`, `CONTENT`, `DELETED`
Status code for error:	`4.03`	`FORBIDDEN`
Response:	payload	JSON Query answer

Table 1: MUSEPA CoAP APIs schema

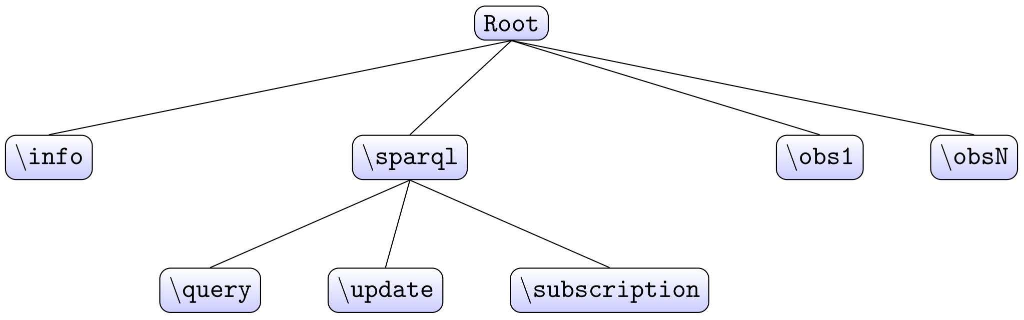

Figure 2: Tree representation of the resources available once MUSEPA is running.

- a `CoAP POST` to `coap://localhost/sparql/update`, providing a SPARQL 1.1 Update payload, to modify the contents of the knowledge base. Additionally, it would be possible to provide the payload in a Turtle format, appending the `?format=ttl` option (this feature is not available in C Minor and in SEPA).
- a `CoAP POST` to `coap://localhost/sparql/subscription`, providing a SPARQL Query payload, to request the creation of an observable resource. If the resource already exists, this step can be skipped;
- a `CoAP OBSERVE` to `coap://localhost/obsX`, to start the observation of the resource, i.e., to start receiving notifications.

These last two points about subscriptions are handled in MUSEPA quite differently than in SEPA and C Minor, and deserves some further considerations. Our goal, with this implementation, is to suggest some ideas in order to simplify the concept of subscription. In SEPA architecture the subscription algorithm reveals its complexity when it comes to identify a set of *added* and *removed* bindings. Let us consider a SPARQL subscription $s(\boldsymbol{b})$ where $\boldsymbol{b}$ is a vector of all the variables that should be bound when executing the query. To define if subscription s has to trigger notifications, for each update the SEPA architecture performs

$$\Delta^s_{\text{added}} = \boldsymbol{b}_{\text{after update}} - \boldsymbol{b}_{\text{before update}} \tag{1}$$

$$\Delta^s_{\text{removed}} = \boldsymbol{b}_{\text{before update}} - \boldsymbol{b}_{\text{after update}} \tag{2}$$

where again $\boldsymbol{b}$ are the bindings of the query, once executed.

Whenever both Δ^s_{added} and $\Delta^s_{\text{removed}}$ are empty, SEPA considers that no notification has to be sent for this subscription s. Otherwise, a notification is sent with the contents of these two vectors. Indeed, whenever a large number of triples is included in the knowledge base, and a large number of subscriptions is currently active, the time of calculation for Δ^s_{added} and $\Delta^s_{\text{removed}}$ can be long.

This timing bottleneck lead us to develop in a previous research C Minor, introducing CoAP to reduce SEPA's dependency on TCP towards UDP, but also going a step further by reducing considerably the calculation of these added and removed bindings. In a similar way, MUSEPA does not perform the calculation, but more simply performs $s(\boldsymbol{b})$ before ($s_{\mathrm{B}}(\boldsymbol{b})$) and after ($s_{\mathrm{A}}(\boldsymbol{b})$) the update. If the result is the same, then no notification is triggered. Otherwise, yes. Within this paper, so, we keep supporting this position: while providing differential evolution of the knowledge base is really a consistent and important feature of SEPA, we argue that such calculation may be too cumbersome to deal with thousands of devices (which is not an impossible IoT scenario nowadays).

MUSEPA, consequently, implements the subscription in mechanism in the aforementioned way. To be more precise the sequence would be the following: (i) the client performs the subscription request; (ii) MUSEPA accepts (or denies) the request and creates an observable resource; also, it provides the path to reach this resource; (iii) the client observes the resource.

A relevant difference between the behavior of C Minor and MUSEPA must be highlighted here, regarding this sequence. C Minor would join point (ii) and (iii) in the apis: the client is there requested to provide the SPARQL query that he/she is interested in, and the name of the observable resource. Leveraging an example from [77], an example of SEPA-C Minor subscription request could be:

```
{
   "query":"SELECT * WHERE { ?s ?p ?o }",
   "alias":"all"
}
```

that would imply the creation of an observable resource at `coap://localhost/all`. We found that C Minor was treating this way each request, denying the ones that were colliding with pre-existing *aliases*. However, the real problem was that there was no way in making two clients observe the same resource, even if they were asking for the same SPARQL query.

To address such problem, we changed the logic in MUSEPA, keeping separated the subscription request from the observation. The subscription request would hereby contain just the SPARQL query, and the payload of the response would contain the hash of the query. MUSEPA, in fact, provides the *alias* to the client (and not *vice versa*) in the form of the hash of the request and creates accordingly the observable resource (represented in Fig. 2 by `\obs1` and `\obsN`)[6]. This means that:

1. two different clients making the very same subscription would end up in observing the same resource. Indeed, one can argue that `SELECT * WHERE {?a ?b ?c}` and `SELECT * WHERE {?s ?p ?o}` *are the same* subscription, but yet they do not have the same hash. Indeed, this is true, but such identity due to the SPARQL language can be easily restored with a few checks that are not in the scope of this paper.
2. from point (1), less resources are used in the server to instantiate the observable entities, and to trigger the notifications. This feature is particularly interesting in IoT environments, where generally we have a high number of devices making extremely similar requests.

An additional problem of this setup is that, as it is known, it is hard, but yet generally possible to find two strings with the same hash. Consequently, theoretically, we may encounter a bug for which two different SPARQL queries have the same hash, and therefore are both directed to the same resource by MUSEPA engine. In our development we considered to the best of our knowledge this probability to be extremely low given the fact that we are requesting that not only the two strings simultaneously used have the same hash, but also that they are *valid* SPARQL queries.

Once the subscription request has been handled, the client is free to observe the given newly created resource.

Some additional features available in MUSEPA:

- The `\info` resource, that provides useful information to the client;
- The possibility to perform `GET` requests to the `\subscription` resource, to get some stats about currently active subscriptions (e.g., the SPARQL query, the number of observing clients);
- The prefix setup. As is it known, both in SPARQL and in Turtle the preamble with the list of prefixes is an important part of the contents. The prefix setup in MUSEPA is a useful feature that allows the clients to avoid sending with each request a list of prefixes. In this way, we simply give MUSEPA the prefixes that we think will be mostly used during our application, and they will be systematically appended to all incoming requests.

Please refer to Section 4 for the evaluation and further discussion on memory/complexity issues.

3.3. Musical Things on the Semantic Web

According to W3C perspective, a Thing can indicate both a physical or a virtual device and each Thing is associated to a Thing Description. This is a sequence of meta-data modeling the interaction patterns (Properties,

[6] in this case `coap://localhost/294a2eefbe9b4220451664e59aaf1c6b` since the hash of "`SELECT * WHERE {?a ?b ?c}`" is `294a2eefbe9b4220451664e59aaf1c6b`

Actions, and Events) as well as the security and access protocol information. Moreover, the Thing Descriptions are intended to be encoded in a standard RDF serialization, e.g. JSON-LD, and therefore in a machine-understandable way.

The most recent recommendations for the Web of Things released by the W3C in June 2021[7] are slightly different from the view presented in this paper. This is due to the fact that, as already said in Section 2.1, one of the origins of this paper is located in [37]. While such work is based on a previous version of W3C drafts, it provides an ontology that was designed to be used by SEPA and its successors, like MUSEPA. As it is stated there, however, the two visions not only can coexist, but will be fully integrated in a future work.

Therefore, following the W3C's WoT vision of an IoT fully integrated with the Web, we equip, where possible, Musical Things with a small HTTP server. The purpose of the server is that of exposing on the Web the Musical Thing Description. Such a description of the device relies on the semantics specified by the ontologies in which MUSEPA operates (i.e., the SWOT, IoMusT, and SMI ontologies). Each Musical Thing may be associated to a public IP address. Devices accessing the Musical Thing Descriptor will find all the necessary information about how to interact and control it.

Nevertheless, not all Musical Things may have the necessary computational resources to handle an HTTP server. This is especially the case of those embedded systems dedicated to real-time audio processing tasks which need to primarily ensure a good audio quality without audio dropouts. In these cases the Musical Thing's description requests can be handled by MUSEPA as follows.

In general, having an HTTP server exposing the Thing Description is just a solution to publicly share it. Given the fact that all information about the environment should be stored in the knowledge base, it is also possible to retrieve a Thing Description by performing the appropriate SPARQL query to the knowledge base directly. An example of Thing Description in an RDF-turtle format can be found in Listing 2 So, alternatives to the HTTP Server:

- Issue a query similar to the one in Listing 1 to MUSEPA, in which we look for Interaction Patterns (Actions, Events, Properties) that belong to a Thing located at `<http://smartguitar.eu>`;

- If the knowledge base endpoint is known and reachable, issue a query similar to the one in Listing 1 to the endpoint (e.g., Fuseki, Blazegraph).

[7] `https://www.w3.org/TR/wot-thing-description/`

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-
    syntax-ns#>
PREFIX swot: <http://purl.org/ontology/swot#>

SELECT * WHERE {
    <http://smartguitar.eu> rdf:type swot:Thing;
            swot:hasThingDescription ?td;
            swot:hasName ?tName.
    OPTIONAL { <http://smartguitar.eu> swot:
    hasSubThing ?subThing } .
    ?td swot:hasInteractionPattern ?ip.
    ?ip rdf:type ?ip_type.
    OPTIONAL { ?ip swot:forProperty ?pTarget } .
    FILTER ( ?ip_type != swot:InteractionPattern
    )
}
```

Listing 1: Example of possible Thing Description discovery query, inspired from [37]

4. Evaluation

This section describes the evaluation of MUSEPA focusing on its key aspects, namely interoperability, discoverability, automatic real-time inference, and latency communication performances. To assess the quality of these features supported by MUSEPA we created a real IoMusT ecosystem around it, and we implemented different tasks accomplished by the involved Musical Things exemplifying possible interactions across different stakeholders. Specifically, we conducted the testing on the most complex among the IoMusT scenarios, namely that of live performance.

The IoMusT ecosystem was composed by a server hosting MUSEPA and the following heterogeneous Musical Things:

- A smart mandolin prototype built around the Bela board (reported in [65]);

- A smart guitar prototype built around a Elk-Pi board (adapting the instrument reported in [7]);

- A shoe-based prototype of musical haptic wearable for a live sound engineer (reported in [12]);

- Two armband-based prototypes of musical haptic wearable for audience members (reported in [13]);

- Three ad-hoc built apps for iOS-based smartphones coded in swift and integrating CoAP;

- An Oculus Quest virtual reality headset, with an engine coded in Unity 3D and integrating CoAP.

The server was an HP Pavillion i7 laptop running Ubuntu Linux 20.04. Connectivity was implemented using the IEEE 802.11ac Wi-Fi standard over the 5GHz band thanks to a TP-Link TL-WR902AC router. Following the recommendations reported in [83] to optimize the components of a Wi-Fi system for live performance scenarios to reduce latency and increase throughput, the router was

```
@prefix swot:       <http://purl.org/ontology/swot#>.
@prefix iomust:     <http://purl.org/ontology/iomust/internet_of_things/iomust#>.
@prefix smi:        <http://purl.org/ontology/iomust/smi#>.
@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/main>.
@prefix foaf:       <http://xmlns.com/foaf/0.1/>.
@prefix xsd:        <http://www.w3.org/2001/XMLSchema#>.
@prefix rdf:        <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.

<http://XSDstringDataSchema.org>    rdf:type                       swot:DataSchema.
<http://XSDdoubleDataSchema.org>    rdf:type                       swot:DataSchema.
<http://XSDintegerDataSchema.org>   rdf:type                       swot:DataSchema.

<http://smartguitar.eu>             rdf:type                       swot:Thing;
                                    rdf:type                       iomust:SmartInstrument;
                                    rdf:type                       sosa:Platform;
                                    swot:hasName                   'Smart_Guitar';
                                    swot:hasThingDescription       <http://smartguitarDescription.eu>.

<http://smartguitarDescription.eu>  rdf:type                       swot:ThingDescription;
                                    swot:hasInteractionPattern     <http://smartguitar.eu/ProvideBatteryValue>,
                                                                   <http://smartguitar.eu/IsBeingPlayed>,
                                                                   <http://smartguitar.eu/TriggerLowBattery>,
                                                                   <http://smartguitar.eu/Battery>;
                                    swot:hasAction                 <http://smartguitar.eu/ProvideBatteryValue>;
                                    swot:hasEvent                  <http://smartguitar.eu/IsBeingPlayed>,
                                                                   <http://smartguitar.eu/TriggerLowBattery>;
                                    swot:hasProperty               <http://smartguitar.eu/Battery>,
                                                                   <http://smartguitar.eu/SoundEngine>,
                                                                   <http://smartguitar.eu/SoftwareMixer>.

<http://smartguitar.eu/ProvideBatteryValue> rdf:type               swot:InteractionPattern, swot:Action;
                                    swot:hasName                   'Smart_Guitar_Provide_Battery_Value';
                                    swot:hasDataSchema             <http://XSDstringDataSchema.org>,
                                                                   <http://XSDdoubleDataSchema.org>;
                                    swot:hasInputDataSchema        <http://XSDstringDataSchema.org>;
                                    swot:hasOutputDataSchema       <http://XSDdoubleDataSchema.org>.


<http://smartguitar.eu/IsBeingPlayed>       rdf:type               swot:InteractionPattern, swot:Event;
                                    swot:hasName                   'Smart_Guitar_Is_Being_Played';
                                    swot:hasOutputDataSchema       <http://XSDbooleanDataSchema.org>.

<http://smartguitar.eu/TriggerLowBattery>   rdf:type               swot:InteractionPattern, swot:Event;
                                    swot:hasName                   'Smart_Guitar_Triggers_Low_Battery';
                                    swot:hasOutputDataSchema       <http://XSDstringDataSchema.org>.

<http://smartguitar.eu/Battery>     rdf:type                       swot:InteractionPattern, swot:Property;
                                    swot:hasName                   'Smart_Guitar_Battery';
                                    swot:isWritable                'false';
                                    swot:hasPropertyDataSchema     <http://XSDdoubleDataSchema.org>;
                                    swot:hasData                   <http://smartguitar.eu/Battery/PropertyData>;
                                    swot:hasPropertyData           <http://smartguitar.eu/Battery/PropertyData>.

<http://smartguitar.eu/SoundEngine>  smi:works_at_audio_sampling_rate "48000"^^xsd:integer .
```

Listing 2: Portion of the Turtle file of the Thing Description of a Smart Guitar.

configured in access point mode, security was disabled, and only the IEEE 802.11ac standard was supported. The Musical Things composing the ecosystem were chosen for their heterogeneity of purposes and system architectures. The purposes ranged from music generation (accomplished by the instruments for performers), haptic stimulation (accomplished by the musical haptic wearables for live sound engineer and audience members), virtual environment rendering (accomplished by the headset for audience members), and screen-based interactions (accomplished by the smartphones for audience members). The architectures spanned from different linux-based hard real-time platforms to iOS and Android-based systems. The integration of CoAP in each of these systems was feasible given the availability of CoAP libraries in different programming languages (swift, python, c#).

4.1. Interoperability and discoverability assessment

First of all, it is necessary to better explain the concept of *discoverability* that is intended to be the focus of this Section. The idea of discoverability can be interpreted at connection level, i.e., the possibility to perform some kind of action to know, as a result, the IP address of each entity connected to the network. Or, conversely, as a kind of discoverability based on higher level features, namely investigating the available devices that could be called to perform some actions, in order to achieve a goal. In this study we focus on the second alternative, since the usage of RDF technologies and MUSEPA gives the opportunity to represent the features of the devices in an interoperable manner. In such a design, devices need to know only the IP address of MUSEPA to discover their environment by means of queries similar to the one reported in Listing 1, which is not the case of the former approach. Nevertheless, it is worth noticing that the two approaches are not necessarily mutually exclusive: the possibility of sharing a connectivity mediated by MUSEPA, and a direct one as in plain IoT technologies, may lead to new facets of the Web of Things which could be investigated in future research.

Interoperability and discoverability are features that arise from the adopted design choices. To be more precise, it is worth mentioning that the adopted SWOT ontology, which is responsible for the dynamic reconfiguration of the environment, has already been evaluated in [37]. Similarly, the IoMusT and SMI ontologies (responsible for semantic interoperability and discoverability) have already been evaluated in their respective studies [17, 11]. As the three ontologies were already evaluated and are available online along with their full documentation[8], it was not necessary to perform an evaluation of them in the present work.

Interoperability is ensured by the use of standards of the Semantic Web, including OWL and turtle. Being shared among devices, they participate to the setup of a Musical IoT semantic environment in which all devices are described according to the same vocabularies. The ontologies utilized are very well documented and accessible. At a device interaction level, interoperability is also guaranteed by the adoption of a protocol, CoAP, which is well supported in all the most widespread programming languages. Discoverability is achieved by means of the dynamic behavior of MUSEPA, which is based on updates, queries, and subscriptions.

A common requirement for IoMusT performance ecosystems is that participants can spontaneously join an experience and interact within a distributed environment composed of different devices. As a proof of concept to demonstrate the interoperability and discoverability aspects, we implemented a performance scenario in which each Musical Thing joins and leaves the network at a different time, and when this happens the behavior of the other Musical Things present in the ecosystem changes. Specifically,

- When the smart mandolin joins the network the two armband-based musical haptic wearables configure themselves to trigger a vibration in response to each note played (see [13]) and the virtual reality headset configures itself to display a virtual environment whose objects are controlled by the sensors of the instrument (see [10]);
- When the guitarist presses a button of the smart guitar, the shoe-based musical haptic wearable worn by the live sound engineer triggers a vibration pattern (e.g., the case of a guitarist willing to call the attention of the live sound engineer, see [12]);
- When the battery level of the smart mandolin gets lower than the 20%, the shoe-based musical haptic wearable worn by the live sound engineer triggers a vibration pattern;
- When a smartphone joins or leaves the ecosystem the other smartphones configure themselves to produce a different musical note.

Listing 3 shows an example of the SPARQL code performing a smart discovery.

4.2. Automatic real-time inference

The possibility to conduct automatically and in real-time a reasoning process on a database containing information about the performance ecosystem represents a novel opportunity for musical applications. To show an example of automatic inference behavior we considered a participatory live music scenario and implemented the following use cases:

- The audience members use their smartphones to send (to a central server hosting MUSEPA) their preference among three choices; a voting mechanism is in

[8] `https://w3id.org/iomust#`
`https://fr4ncidir.github.io/SemanticWoT/`
`https://w3id.org/smi#`

```
SELECT * WHERE {
  ?a rdf:type swot:Thing, iomust:SmartInstrument; swot:hasName 'Smart_Guitar' .
  ?a swot:hasThingDescription ?td .
  ?td swot:hasAction ?action .
  ?action swot:hasDataSchema ?dt .
  ?action ?dataschematype ?dt .
}"
```

Listing 3: SPARQL example of smart discovery of the possible actions provided by a smart guitar, including the actions' types of inputs and outputs.

place on the server such that if at least two audience members performed the same choice then the smart mandolin and the smart guitar are notified and parameters of their sound engine (i.e., a delay feedback and a reverb time) are automatically changed to produce a different timbre;

- All times that the smart guitar and the smart mandolin play together, the smartphones blink with a yellow light and the VR headset displays a yellow-colored scenes; when the smart guitar is playing and the smart mandolin is not playing, then the smartphones blink with a blue light and the VR headset displays a blue-colored scenes; when the smart mandolin is playing and the smart guitar is not playing, then the smartphones blink with a red light and the VR headset displays a red-colored scenes;

A relevant concept that should be considered from this setup, and that can be also applied to other environments, is that while the environment works as a whole to realize the application, MUSEPA also provides the support for reconfiguration (i.e., reconfiguration knowledge-based). A device being removed would trigger the notifications to the subscribed entities, if any, which could thus take countermeasures, if needed

4.3. Latency assessment

As stated earlier, while implementing MUSEPA we attempted to be careful with resource management. As a result, we introduced the hash-naming of the observable resources, to avoid multiple instances for the same subscription, as well as the prefix control to reduce the amount of data exchanged. In this Subsection we provide some numerical insights on this work. Before proceeding, however, a few more points need to be highlighted.

Which evaluation is reasonable for our architecture? Indeed, we know that CoAP runs on UDP and, therefore, the reliability of packets reaching destination is a factor that is dependent on the network, which is far beyond the scope of this paper. Therefore, this is not a valid starting point.

As a matter of fact, we can consider that any call to MUSEPA implies a global execution time T_{request}

$$T_{\text{request}} = \tau_{\text{CtoM}} + t_{\text{M}} + \tau_{\text{MtoC}} \tag{3}$$

where τ_{CtoM} is the time that a request takes to travel from the client C to MUSEPA (CtoM); vice versa, τ_{MtoC} is the time from MUSEPA to the client (MtoC). Then, t_{M} is the elaboration time within MUSEPA itself.

It is clear from Eq. 3 that τ_{MtoC} and τ_{CtoM} are the aforementioned timings that depend on plenty of factors unrelated with how the MUSEPA internal engine is programmed, which is not the case of t_{M}. In this Section, consequently, we will focus exclusively on t_{M}, with some caveats:

- For Queries, the time $t_{\text{M}} = t_{\text{Query}}$, that is the difference from the time at which the query request is fulfilled from the time at which it is received. This is an evaluation that depends completely on the endpoint chosen, RDFLib, Blazegraph, Fuseki or others, and therefore we intend it as not relevant for our research, since the topic is not to perform an evaluation of the endpoint's performances;

- Update evaluation can be modeled as a problem similar to query: i.e., the time $t_{\text{M}} = t_{\text{Update}}$ is the difference between the time at which update is committed and the time at which the update request is received, with additional information about the format that can be SPARQL 1.1 payload update and Turtle payload update. Again, this is also an evaluation that depends completely on the endpoint chosen, and therefore we intend it out of the scope of the paper. The real interest of this research is in the timings that bound notification performances (see next point), that is a complex compound of update and query timings;

- For Subscriptions, conversely, the time $t_{\text{M}} = t_{\text{Notification}}$ is the time elapsed from the moment in which an update is received up to the moment in which *the last client subscribed* is notified (see Fig. 3). This is the value that we consider mostly relevant to obtain some numerical evaluation of MUSEPA's architecture performance boundaries, since the full execution of the notification engine is included in this time measurement.

On a practical point of view, the evaluation was implemented as follows. First of all, we refer from now on to a MUSEPA server running over an RDFLib triple store. The reason for this choice is that we actually want to make an evaluation targeting possible IoMusT applications in a realistic way (which would not be the case, as explained in the previous Sections, with Blazegraph or Fuseki). Other triple stores (Fuseki, Blazegraph...) all introduce an HTTP protocol communication in the process, which is definitely not a good choice (due to protocol design) for musical applications. Although it is clearly important to cite them, it appears to the authors that a full comparison would have been rather irrelevant to the musical environment: even the simplest case would have been not acceptable in terms of performances.

Secondly, we obtained a set of test cases for a relatively simple scenario designed to represent a situation in which MUSEPA must handle larger and larger payloads in notification bodies. With reference to IoT and IoMust, large payloads can be considered as "bad" cases, since usually for IoT and IoMusT we would rather expect smaller payloads, but frequent notifications and a large number of clients. So, the two main evaluation directions to be followed, then, are (i) testing with an increasing number of triples exchanged; (ii) testing with an increasing number of subscribers.

To better explain these two choices, please consider what follows. We hereby target IoT and, more specifically, IoMusT environments, which we try to implement as an integration of devices. Modern IoT pervasiveness, in particular, allows us to expect a larger and larger number of similar devices, all performing similar subscriptions within their own context. In addition to that, we can also expect devices that are constrained in their hardware, memory, bandwidth and that will unlikely perform extremely complex operations or edge data integration. These two factors, eventually, support the choice of (i), realizing a scenario with a lot of triples being shared; and (ii), stressing the idea of a large number of actors in the environment.

The number of triples, as it can be seen in Fig. 3, spans from 1 up to 40000. They have been obtained by filling the RDFLib knowledge base with the result of querying DBpedia[9] with a simple `SELECT * WHERE {?a ?b ?c} LIMIT %i`, where `%i` is a constant related to the number of the triples of the test.

The number of subscribers waiting for the aforementioned 1 to 40000 triples update notification, in the other hand, was tested in three different cases: 10 clients, 25 clients and 40 clients. They all do as a concept the same subscription `SELECT * WHERE {?A ?B ?C}`, even though we took care of creating random variable names. This helped us realizing a test that takes into account non overlapping subscriptions (see Section 3.2, where we describe the hashed-alias mechanism). Indeed, even the 40 simulated clients is not much, if we consider a general IoT and IoMusT environment: however, since MUSEPA is a prototype, and future works are intended to address real implementations of IoMusT environments, we consider that the present research can be a relevant starting point for further enhancements not only in IoMusT and possibly general IoT, but also in the implementation of MUSEPA for real world applications. The code can be found in the same GitHub repository of the project.

We present here a result of update-to-notification measurement in a graphical form in Fig. 3. This measurement has been made on a Windows™10 (Ubuntu subsystem) 20.04.1 64 bits running on an Intel® Core™ i7-9750H CPU @ 2.60 GHz with 16 GB of RAM memory. As it can be seen in the figure, the notification time measure is divided into two charts: one where the time elapsed between the update request and first notification is reported, and the other where there is the time elapsed between the update request and last notification. What we can understand from the charts, overall, is that MUSEPA suffers like SEPA of the problem of rapidly increasing notification time along with the increasing number of subscribing clients and triples, although the minimum notification time appears to be dependent only on the number of triples (which is a direct consequence of the usage of `aiocoap` library in MUSEPA code). Let us mention again that the RDF triples endpoint is here fixed to be RDFLib. The timings, anyway, clearly show that yet MUSEPA cannot deal with large IoMusT setups, since we have performances that, as of now, are definitely not sufficient for musical applications that need a large number of triples (i.e., a large number of connected objects interacting). At least, the results appear to be subjected to a linear growth in our test case, compared to the tests that were done for SEPA in [20], where performances were superlinear on number of triples in the knowledge base and on the number of subscribers, due to the complexity of the binding difference calculation algorithm. This is an encouraging result, although there is clearly room for more work and more enhancements in the future, if we consider the target of making a real semantic IoMusT application.

Nevertheless, given the fact that MUSEPA is here presented as a prototype in Python, we are confident that a C++ implementation exploiting the complete features of multithreading would reasonably achieve better results over a less generic test.

5. Discussion

Important advantages of our system include i) the openness of the proposed architecture, which is ensured by the adoption of Semantic Web standards; ii) the separation of the logical data model from implementation details. This makes the data models, representations and their relation to musical concepts, events or actions reusable and improves interoperability overall. The graph-based conceptualization of RDF data representation also lends itself to representing complex musical metadata more easily compared to tree-based structures such as XML or pure JSON or protocols that only support ad-hoc semantics such as OSC. The benefits of this is apparent in how MUSEPA is

[9] `https://www.dbpedia.org/`

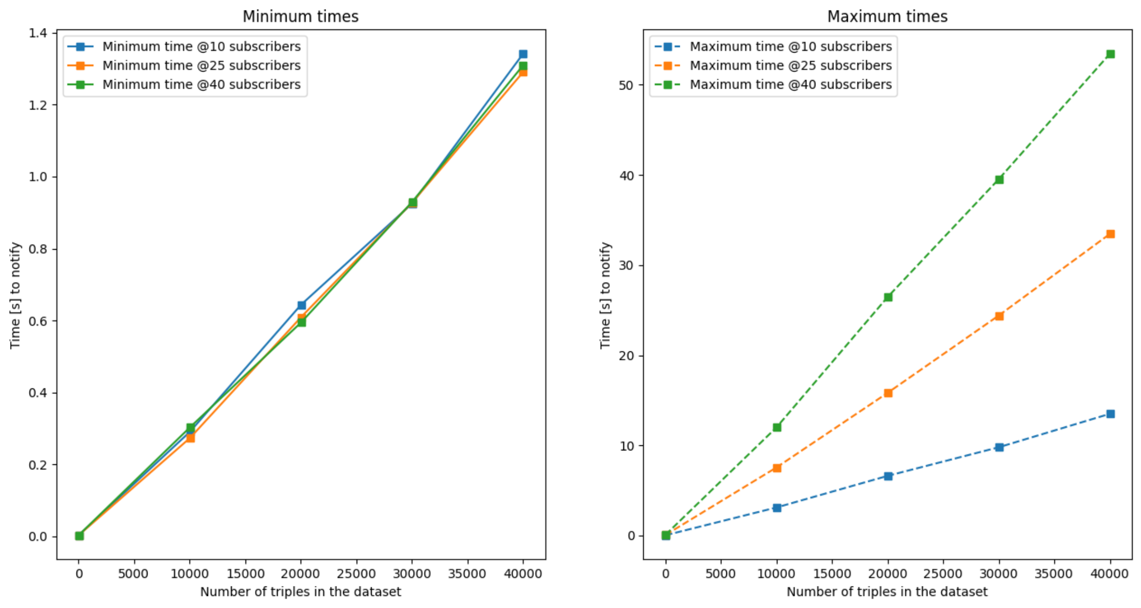

Figure 3: Update to notification time measurement

easily able to serve as an arbitrator in a musical performance environment, which is the most complex among the IoMusT scenarios.

Notably, MUSEPA was not devised to support continuous real-time interactions across devices either locally or remotely. The latencies obtainable even with very efficient implementations of the architecture would be anyways too high to ensure perceptually-valid control by a human musician. Rather, MUSEPA's design was based on goals such as the support for the fast discovery of when a new Musical Thing joins the ecosystem or the fast automatic reconfiguration of a complex ecosystem based on inference mechanisms. These are discrete interactions. For continuous real-time interactions between Musical Things, protocols such as MIDI or OSC are a better choice. Nevertheless, nothing prevents the use of such protocols in conjunction with MUSEPA in order to take advantage of their complementary benefits within a same ecosystem. Furthermore, in the current study we did not introduce any semantic reasoning engines in the MUSEPA architecture. We plan to investigate such a possible extension in the near future.

MUSEPA was conceived to overcome the limitations of existing musical communication protocols with the aim of addressing a better communication within IoMusT ecosystems. Therefore, it is worth comparing MUSEPA with libmapper, being this the system currently closest to MUSEPA within the music technology domain. Notably, to be sensible in the context of the present study this comparison needs to be made in relation to the IoMusT field. Firstly, libmapper was conceived to overcome some limitations of the OSC protocol. Whereas flexibility in development is provided by the fact that OSC messages are tagged with a user-specifiable, human-readable string instead of a predetermined controller ID number, this has the drawback to hinder interoperabilty as devices do not communicate with a common language. libmapper approaches this drawback by providing a mechanism of translation from an OSC description into another. Conversely, MUSEPA does not perform any translation. Devices are required to adopt the language described by the utilized ontologies. Secondly, whereas libmapper simply provides a minimal layer to enable devices describing themselves and their capabilities, MUSEPA adopts the full expressive power of ontological representation, which has also the benefit to enable automatic reasoning processes over the knowledge base. Thirdly, libmapper was not conceived for interactions across the Internet. Conversely, MUSEPA can support both local and remote interactions across devices. Along the same lines, libmapper was not designed for supporting the integration with the Web, which is instead a feature fully supported in MUSEPA. Fourthly, libmapper was conceived for supporting a continuous and parametric control where a controller present in a system (e.g., a sensor) directly controls the parameter/s of another system (e.g., a parameter of a synthesizer). MUSEPA adopts a completely different approach, which is based on a publish/subscribe mechanism. Nevertheless, continuous parametric control is also supported, although this approach is sub-optimal from the latency standpoint. Furthermore, given the adopted distributed approach, to record and store data about the musical session (i.e., the exchanged messages) libmapper needs to recur to a central hub additional to the devices communicating. This solution, however, requires the doubling of all messages occurring in the ecosystem which can cause large amounts of traffic as the number of nodes increases. MUSEPA by using a centralized network topology is exempt from these issues.

Whereas, all these features make MUSEPA an ideal candidate to support communication in IoMusT ecosystems, the current implementation of MUSEPA presents also some limitations. Firstly, the communication latency may be improved. However, this is likely a feature of the current implementation which relies on Python. A C++ implementation is expected to drastically improve the processing latencies reported in Section 4.3. Secondly, at present, a graphical user interface is missing, which would help less technically skilled musicians to configure their Musical Things and leverage the interaction capabilities afforded by the ecosystem. A dashboard could facilitate users to interact with each Musical Thing by visualizing its properties, executing commands or observing the notifications produced (an approach followed in [84]). Future research is needed to understand the usability of the system by means of extensive user testing. Thirdly, developing ecosystems with the proposed framework may be more complex than with non-semantic approaches. The development effort may not pay back for small-scale ecosystems with homogeneous devices and limited application scope. Conversely, the presented approach is beneficial in dynamic complex scenarios like large-scale distributed performances with several stakeholders, where wide interoperability is required and sensing and actuating tasks need careful selection of data sources and devices. Furthermore, a current limitation of the developed system is the lack of security support. Security is a crucial issue for the IoMusT domain [1] as well as for MUSEPA, given the possibility offered by our architecture to discover, interact, and potentially update remote Musical Things. Therefore, proper authorization mechanisms must be designed to avoid data leaks or harmful operations on shared resources within IoMusT leveraging MUSEPA.

6. Conclusions

In this paper, we proposed the integration of the Semantic Web in the Web of Things within the musical domain, leading to the Semantic Web of Musical Things paradigm. We presented MUSEPA, a semantic publish/-subscribe broker specifically designed to meet the requirements of Internet of Musical Things ecosystems. Specifically, MUSEPA adapts the concept of a SPARQL Event Processing Architecture [20] to CoAP, a lightweight application protocol commonly adopted in the IoT. The evaluation showed the ability of the developed architecture in acting as an interoperability enabler allowing multiple heterogeneous Musical Things to cooperate, relying on shared music-related ontologies. Besides automatic discovery, the architecture is able to support communications with communication latencies acceptable for most of the musical activities envisioned in the IoMusT.

In future works we plan to enhance MUSEPA in different ways. Firstly, we plan to port the current Python implementation to C++. This is expected to lead to much better performances in terms of processing latency. Secondly, we will study how it would be possible to make further advancements in the field of Web of Things and Musical Web of Things by exploring the current recommendations of W3C concerning the Scripting APIs and the Thing Description ontology, that are evolving in parallel with SWOT and the current realization of MUSEPA. Thirdly, we will enhance MUSEPA with security features so that IoMusT forming around such a technology are ensured to be socially desirable and undertaken in the public interest. Fourthly, we plan to create a graphical user interface to facilitate the adoption of MUSEPA also by those less technically skilled. Along the same lines we envision a direct integration of MUSEPA communication mechanisms in software for real-time music processing such as Pure Data or Max/MSP. In addition we will consider the integration of specialized compression algorithms for Semantic Web languages (as reported in [21]) to achieve a further reduction of storage and network load. Furthermore, we will further validate the operations of the presented architecture on large-scale IoMusT deployments.

To date, standardization activities for the IoMusT are mostly unrealized [1] and are crucial for its success and indispensable to avoid the fragmentation that characterizes the general IoT field [85]. The work reported in this paper aimed to perform a decisive step towards this direction.

Ultimately, this study showed that an implementation of Semantic Web of Musical Things makes it possible to create ecosystems supporting various types of interactions across different stakeholders, such as performers and audience members. The application of MUSEPA as a system to exchange information across heterogeneous Musical Things, has the potential to result in novel semantically-based interactions between stakeholders as well as between stakeholders and musical content. These include novel kinds of live music performances, teaching methodologies, and approaches to composition. The authors look forward to future applications of the proposed technology in different IoMusT ecosystems.

Acknowledgments

The authors wish to thank Mr. Jehad Melad for his valuable work and help in implementing and testing MUSEPA functionalities during his internship at École Nationale Supérieure des Mines de Saint-Étienne; and Professors Flavien Balbo and Mihaela Juganaru-Mathieu for their support and advices on the architecture being developed.

References

[1] L. Turchet, C. Fischione, G. Essl, D. Keller, M. Barthet, Internet of Musical Things: Vision and Challenges, IEEE Access 6 (2018) 61994–62017. doi:https://doi.org/10.1109/ACCESS.

2018.2872625.
URL https://doi.org/10.1109/ACCESS.2018.2872625

[2] R. Hupke, M. Nophut, S. Preihs, J. Peissig, Perceptual evaluation of an augmented audience service under realistic live conditions, in: Audio Engineering Society Convention 145, Audio Engineering Society, 2018.

[3] B. Matuszewski, F. Bevilacqua, Toward a Web of Audio Things, in: Proceedings of the Sound and Music Computing Conference, 2018.

[4] D. Keller, C. Gomes, L. Aliel, The handy metaphor: Bimanual, touchless interaction for the internet of musical things, Journal of New Music Research 48 (4) (2019) 385–396.

[5] J. Martinez-Avila, C. Greenhalgh, A. Hazzard, S. Benford, A. Chamberlain, Encumbered interaction: a study of musicians preparing to perform, in: Proceedings of the Conference on Human Factors in Computing Systems, ACM, 2019.

[6] O. Bown, S. Ferguson, L. Bray, A. Fraietta, L. Loke, Facilitating creative exploratory search with multiple networked audio devices using happybrackets, in: Proceedings of the International Conference on New Interfaces for Musical Expression, 2019, pp. 286–291.

[7] L. Turchet, J. Pauwels, C. Fischione, G. Fazekas, Cloud-smart musical instrument interactions: Querying a large music collection with a smart guitar, ACM Transactions on the Internet of Things 1 (3) (2020) 1–29. doi:https://doi.org/10.1145/3377881.
URL https://dl.acm.org/doi/abs/10.1145/3377881

[8] L. Turchet, C. Fischione, Elk Audio OS: an open source operating system for the Internet of Musical Things, ACM Transactions on the Internet of Things 2 (2) (2021) 1–18.

[9] D. Keller, V. Lazzarini, Ecologically grounded creative practices in ubiquitous music, Organised Sound 22 (1) (2017) 61–72. doi:10.1017/S1355771816000340.
URL https://doi.org/10.1017/S1355771816000340

[10] L. Turchet, M. Benincaso, C. Fischione, Examples of use cases with smart instruments, in: Proceedings of Audio Mostly Conference, 2017, pp. 47:1–47:5. doi:10.1145/3123514.3123553.
URL https://doi.org/10.1145/3123514.3123553

[11] L. Turchet, Smart Musical Instruments: vision, design principles, and future directions, IEEE Access 7 (2019) 8944–8963. doi:10.1109/ACCESS.2018.2876891.
URL https://doi.org/10.1109/ACCESS.2018.2876891

[12] L. Turchet, M. Barthet, Co-design of Musical Haptic Wearables for electronic music performer's communication, IEEE Transactions on Human-Machine Systems 49 (2) (2019) 183–193. doi:10.1109/THMS.2018.2885408.

[13] L. Turchet, T. West, M. M. Wanderley, Touching the audience: Musical Haptic Wearables for augmented and participatory live music performances, Journal of Personal and Ubiquitous Computing (2020) 1–21doi:https://doi.org/10.1007/s00779-020-01395-2.
URL https://doi.org/10.1007/s00779-020-01395-2

[14] M. A. Razzaque, M. Milojevic-Jevric, A. Palade, S. Clarke, Middleware for internet of things: a survey, IEEE Internet of things journal 3 (1) (2015) 70–95.

[15] M. Wright, A. Freed, A. Momeni, Opensound control: State of the art 2003, in: Proceedings of the Conference on New Interfaces for Musical Expression, 2003, pp. 153–160.

[16] J. Malloch, S. Sinclair, M. Wanderley, Distributed tools for interactive design of heterogeneous signal networks, Multimedia Tools and Applications 74 (15) (2015) 5683–5707.

[17] L. Turchet, F. Antoniazzi, F. Viola, F. Giunchiglia, G. Fazekas, The internet of musical things ontology, Journal of Web Semantics 60 (2020) 100548. doi:https://doi.org/10.1016/j.websem.2020.100548.
URL http://www.sciencedirect.com/science/article/pii/S1570826820300019

[18] T. Berners-Lee, J. Hendler, O. Lassila, The semantic web, Scientific american 284 (5) (2001) 34–43.

[19] J. Sowa, Knowledge representation: logical, philosophical, and computational foundations, Vol. 13, Brooks/Cole Pacific Grove, CA, 2000.

[20] L. Roffia, P. Azzoni, C. Aguzzi, F. Viola, F. Antoniazzi, T. Salmon Cinotti, Dynamic Linked Data: A SPARQL Event Processing Architecture, Future Internet 10 (4) (2018) 36.

[21] F. Scioscia, M. Ruta, Building a semantic web of things: issues and perspectives in information compression, in: IEEE International Conference on Semantic Computing, IEEE, 2009, pp. 589–594.

[22] A. Rhayem, M. B. A. Mhiri, F. Gargouri, Semantic web technologies for the internet of things: Systematic literature review, Internet of Things (2020) 100206.

[23] D. Guinard, V. Trifa, F. Mattern, E. Wilde, From the internet of things to the web of things: Resource-oriented architecture and best practices, in: Architecting the Internet of things, Springer, 2011, pp. 97–129.

[24] O. Lassila, R. Swick, Resource description framework (rdf) model and syntax specification (1998).
URL http://www.w3.org/TR/REC-rdf-syntax/

[25] D. Brickley, R. Guha, B. McBride, Rdf schema 1.1, W3C recommendation 25 (2014) 2004–2014.

[26] D. McGuinness, F. Van Harmelen, Owl web ontology language overview (2004).
URL https://www.w3.org/TR/owl-features/

[27] A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, S. Harris, K. Idehen, O. Corby, K. Kjernsmo, Sparql/update: A language for updating rdf graphs, Tech. rep., W3C (2008).

[28] E. Prud, A. Seaborne, SPARQL query language for RDF, Tech. rep., W3C (2006).

[29] S. Wyse, L.and Subramanian, The viability of the web browser as a computer music platform, Computer Music Journal 37 (4) (2013) 10–23.

[30] L. Wyse, Spatially distributed sound computing and rendering using the web audio platform, in: Proceedings of the Web Audio Conference, 2015.

[31] S. Robaszkiewicz, N. Schnell, Soundworks–a playground for artists and developers to create collaborative mobile web performances, in: Proceedings of the Web Audio Conference, 2015.

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

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

[34] M. Ulieru, R. Doursat, Emergent engineering: a radical paradigm shift, International Journal of Autonomous and Adaptive Communications Systems 4 (1) (2011) 39.

[35] S. Waters, Performance ecosystems: Ecological approaches to musical interaction, EMS: Electroacoustic Music Studies Network (2007) 1–20.

[36] A. Di Scipio, 'sound is the interface': from interactive to ecosystemic signal processing, Organised Sound 8 (3) (2003) 269–277.

[37] F. Antoniazzi, F. Viola, Building the semantic web of things through a dynamic ontology, IEEE Internet of Things Journal 6 (6) (2019) 10560–10579.

[38] L. Turchet, P. Bouquet, A. Molinari, G. Fazekas, The Smart Musical Instruments Ontology, Journal of Web Semantics (2022) 100687.

[39] A. Kamilaris, F. Gao, F. Prenafeta-Bold, M. Ali, Agri-iot: A semantic framework for internet of things-enabled smart farming applications, in: IEEE World Forum on Internet of Things, IEEE, 2016, pp. 442–447.

[40] F. Viola, F. Antoniazzi, C. Aguzzi, C. Kamienski, L. Roffia, Mapping the ngsi-ld context model on top of a sparql event processing architecture: Implementation guidelines, in: 2019 24th Conference of Open Innovations Association (FRUCT), 2019, pp. 493–501. doi:10.23919/FRUCT.2019.8711888.

[41] D. Puiu, P. Barnaghi, R. Tönjes, D. Kümper, M. Ali, A. Mileo, J. Parreira, M. Fischer, S. Kolozali, N. Farajidavar, Citypulse: Large scale data analytics framework for smart cities, IEEE Access 4 (2016) 1086–1108.

[42] A. Kamilaris, A. Pitsillides, F. Prenafeta-Bold, M. Ali, A web of things based eco-system for urban computing-towards smarter cities, in: International Conference on Telecommunications, IEEE, 2017, pp. 1–7.

[43] R. Zgheib, E. Conchon, R. Bastide, Semantic middleware architectures for iot healthcare applications, in: Enhanced Living Environments, Springer, 2019, pp. 263–294.

[44] G. Marques, N. Garcia, N. Pombo, A survey on iot: architectures, elements, applications, qos, platforms and security concepts, in: Advances in Mobile cloud computing and big data in the 5G era, Springer, 2017, pp. 115–130.

[45] F. Antoniazzi, G. Paolini, L. Roffia, D. Masotti, A. Costanzo, T. S. Cinotti, A web of things approach for indoor position monitoring of elderly and impaired people, in: 2017 21st Conference of Open Innovations Association (FRUCT), IEEE, 2017, pp. 51–56.

[46] V. Charpenay, S. Käbisch, H. Kosch, Introducing thing descriptions and interactions: An ontology for the web of things, in: SR+ SWIT@ ISWC, 2016, pp. 55–66.

[47] F. Serena, M. Poveda-Villalón, R. García-Castro, Semantic discovery in the web of things, in: International Conference on Web Engineering, Springer, 2017, pp. 19–31.

[48] L. Roffia, F. Morandi, J. Kiljander, A. D'Elia, F. Vergari, F. Viola, L. Bononi, T. S. Cinotti, A semantic publish-subscribe architecture for the internet of things, IEEE Internet of Things Journal 3 (6) (2016) 1274–1296.

[49] T. Elsaleh, S. Enshaeifar, R. Rezvani, S. T. Acton, V. Janeiko, M. Bermudez-Edo, Iot-stream: A lightweight ontology for internet of things data streams and its use with data analytics and event detection services, Sensors 20 (4) (2020) 953.

[50] F. Xhafa, B. Kilic, P. Krause, Evaluation of iot stream processing at edge computing layer for semantic data enrichment, Future Generation Computer Systems 105 (2020) 730–736.

[51] M. Endler, J.-P. Briot, F. S. E. Silva, V. P. de Almeida, E. H. Haeusler, An approach for real-time stream reasoning for the internet of things, in: 2017 IEEE 11th International Conference on Semantic Computing (ICSC), IEEE, 2017, pp. 348–353.

[52] Q. Zhou, Y. Simmhan, V. Prasanna, Knowledge-infused and consistent complex event processing over real-time and persistent streams, Future Generation Computer Systems 76 (2017) 391–406.

[53] S. Gillani, A. Zimmermann, G. Picard, F. Laforest, A query language for semantic complex event processing: Syntax, semantics and implementation, Semantic Web 10 (1) (2019) 53–93.

[54] M. Bermudez-Edo, T. Elsaleh, P. Barnaghi, K. Taylor, Iot-lite: a lightweight semantic model for the internet of things and its use with dynamic semantics, Personal and Ubiquitous Computing 21 (3) (2017) 475–487.

[55] A. Haller, K. Janowicz, S. J. Cox, M. Lefrançois, K. Taylor, D. Le Phuoc, J. Lieberman, R. García-Castro, R. Atkinson, C. Stadler, The sosa/ssn ontology: a joint wec and ogc standard specifying the semantics of sensors observations actuation and sampling, in: Semantic Web, Vol. 1, IOS Press, 2018, pp. 1–19.

[56] V. Charpenay, S. Käbisch, H. Kosch, μrdf store: Towards extending the semantic web to embedded devices, in: European Semantic Web Conference, Springer, 2017, pp. 76–80.

[57] T. Käfer, S. R. Bader, L. Heling, R. Manke, A. Harth, Exposing internet of things devices via rest and linked data interfaces, in: Proc. 2nd workshop semantic web technol. Internet Things, 2017, pp. 1–14.

[58] T. Käfer, A. Harth, Specifying, monitoring, and executing workflows in linked data environments, in: International Semantic Web Conference, Springer, 2018, pp. 424–440.

[59] M. Iglesias-Urkia, A. Gómez, D. Casado-Mansilla, A. Urbieta, Automatic generation of web of things servients using thing descriptions, Personal and Ubiquitous Computing (2020) 1–17.

[60] V. Charpenay, S. Käbisch, H. Kosch, Semantic data integration on the web of things, in: Proceedings of the 8th International Conference on the Internet of Things, 2018, pp. 1–8.

[61] E. Meneses, J. Wang, S. Freire, M. M. Wanderley, A comparison of open-source linux frameworks for an augmented musical instrument implementation, in: Proceedings of the Conference on New Interfaces for Musical Expression, 2019, pp. 222–227.

[62] A. McPherson, V. Zappi, An environment for Submillisecond-Latency audio and sensor processing on BeagleBone black, in: Audio Engineering Society Convention 138, Audio Engineering Society, 2015.
URL `http://www.aes.org/e-lib/browse.cfm?elib=17755`

[63] C. Rottondi, C. Chafe, C. Allocchio, A. Sarti, An overview on networked music performance technologies, IEEE Access 4 (2016) 8823–8843. `doi:10.1109/ACCESS.2016.2628440`.
URL `https://doi.org/10.1109/ACCESS.2016.2628440`

[64] L. Gabrielli, S. Squartini, Wireless Networked Music Performance, Springer, 2016. `doi:10.1007/978-981-10-0335-6_5`.
URL `https://doi.org/10.1007/978-981-10-0335-6_5`

[65] L. Turchet, Smart Mandolin: autobiographical design, implementation, use cases, and lessons learned, in: Proceedings of Audio Mostly Conference, 2018, pp. 13:1–13:7. `doi:10.1145/3243274.3243280`.
URL `http://doi.acm.org/10.1145/3243274.3243280`

[66] L. Turchet, A. McPherson, M. Barthet, Real-time hit classification in a Smart Cajón, Frontiers in ICT 5 (16). `doi:10.3389/fict.2018.00016`.
URL `https://doi.org/10.3389/fict.2018.00016`

[67] L. Turchet, M. Barthet, An ubiquitous smart guitar system for collaborative musical practice, Journal of New Music Research 48 (4) (2019) 352–365. `doi:10.1080/09298215.2019.1637439`.
URL `http://dx.doi.org/10.1080/09298215.2019.1637439`

[68] S. Papetti, C. Saitis (Eds.), Musical Haptics, Springer Series on Touch and Haptic Systems., Springer, Cham, 2018. `doi:10.1007/978-3-319-58316-7`.
URL `https://doi.org/10.1007/978-3-319-58316-7`

[69] O. Hödl, G. Fitzpatrick, F. Kayali, Design implications for technology-mediated audience participation in live music, in: Proceedings of the Sound and Music Computing Conference, 2017, pp. 28–34.

[70] E. Miranda, M. Wanderley, New digital musical instruments: control and interaction beyond the keyboard, Vol. 21, AR Editions, Inc, 2006.

[71] A. Fraietta, Open sound control: Constraints and limitations, in: Proceedings of the Conference on New Interfaces for Musical Expression, 2008, pp. 19–23.

[72] J. Malloch, S. Sinclair, M. Wanderley, A network-based framework for collaborative development and performance of digital musical instruments, in: Computer Music Modeling and Retrieval. Sense of Sounds, Springer Berlin Heidelberg, 2008, pp. 401–425.

[73] J. Malloch, S. Sinclair, M. Wanderley, Libmapper: (a library for connecting things), in: Extended Abstracts on Human Factors in Computing Systems, ACM, 2013, pp. 3087–3090. `doi:10.1145/2468356.2479617`.

[74] M. Baalman, H. Smoak, C. Salter, J. Malloch, M. Wanderley, Sharing data in collaborative, interactive performances: the senseworld datanetwork, in: Proceedings of the Conference on New Interfaces for Musical Expression, 2009, pp. 131–134.

[75] L. Turchet, F. Viola, G. Fazekas, M. Barthet, Towards a Semantic Architecture for Internet of Musical Things applications, in: IEEE Conference of Open Innovations Association (FRUCT), IEEE, 2018, pp. 382–390. `doi:10.23919/FRUCT.2018.8587917`.

[76] T. Lebo, S. Sahoo, D. McGuinness, K. Belhajjame, J. Cheney, D. Corsar, D. Garijo, S. Soiland-Reyes, S. Zednik, J. Zhao, Prov-o: The prov ontology, W3C recommendation 30.

[77] F. Viola, L. Turchet, G. Antoniazzi, F. Fazekas, C Minor: a Semantic Publish/Subscribe Broker for the Internet of Musical Things, in: IEEE Conference of Open Innovations Association (FRUCT), IEEE, 2018, pp. 405–415. `doi:10.23919/FRUCT.2018.8588087`.
URL `https://ieeexplore.ieee.org/document/8588087`

[78] Z. Shelby, K. Hartke, C. Bormann, The constrained application protocol (coap), Tech. rep. (2014).

[79] C. Bormann, A. Castellani, Z. Shelby, Coap: An application protocol for billions of tiny internet nodes, IEEE Internet Com-

puting 16 (2) (2012) 62–67.

[80] N. Naik, Choice of effective messaging protocols for iot systems: Mqtt, coap, amqp and http, in: IEEE International Systems Engineering Symposium, IEEE, 2017, pp. 1–7.

[81] D. Guinard, V. Trifa, Building the web of things: with examples in node. js and raspberry pi, Manning Publications Co., 2016.

[82] L. Feigenbaum, G. T. Williams, K. G. Clark, E. Torres, Sparql 1.1 protocol, Tech. rep., W3C (2013).

[83] T. Mitchell, S. Madgwick, S. Rankine, G. Hilton, A. Freed, A. Nix, Making the most of wi-fi: Optimisations for robust wireless live music performance, in: Proceedings of the Conference on New Interfaces for Musical Expression, 2014, pp. 251–256.

[84] L. Sciullo, L. Gigli, A. Trotta, M. Di Felice, Wot store: Managing resources and applications on the web of things, Internet of Things 9 (2020) 100164.

[85] E. Borgia, The Internet of Things vision: Key features, applications and open issues, Computer Communications 54 (2014) 1–31.